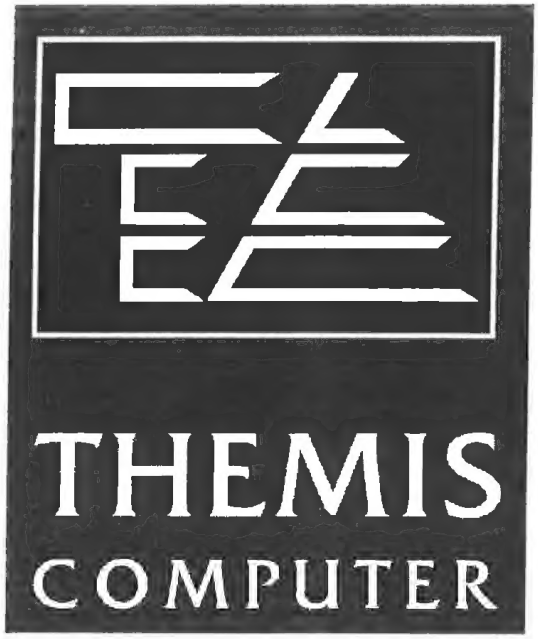


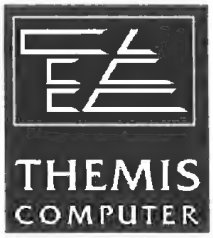
USP-I

Programmer's Guide

Part Number: 104895

USP-1 Programmer's Guide

Revision A1.0 —July 14, 1998



Themis Computer—Americas and Pacific Rim
3185 Laurelview Court
Fremont, CA 94538
Phone (510) 252-0870
Fax (510) 490-5529
World Wide Web http://www.themis.com

Themis Computer—Rest of World
1, Rue Des Essarts
Z.A. De Mayencin
38610 Gieres, France
Phone 33 76 59 60 61
Fax 33 76 63 00 30

Themis Customer Support

North America, South America, and Pacific Rim

Telephone: 510-252-0870
Fax: 510-490-5529
E-mail: support@themis.com
Web Site: http://www.themis.com

USP-1 Programmer's Guide
July 1998
Part Number: 104895

Version Revision History

Revision A **July 14, 1998**

Release of Document to operations.

Corrected several typographical mistakes in tables.

Updated Section 3.5.3, "Accessing the VMEbus from OBP."

Engineering Revision X1.3 **May 28, 1998**

Eliminated the word "Preliminary".

Engineering Revision X1.2 **September 29, 1997**

Engineering Revision X1.1 **September 25, 1997**

Added ThemisVME and Device Driver Sections to Programmers Guide Chapter.

Engineering Revision X1.0 **September 22, 1997**

Version 1.0 **December 1996**

Changes: Initial Release of Document

Table of Contents

List of Tables

List of Figures

1 Introduction

1.1 Overview

Thank-you for purchasing the Themis USP-1 single board computer with VME interface. Themis computer is a leading manufacturer of SPARC based processor boards for the VMbus market. We value our customers comments and concerns, are eager to know what you think of our product, and hear any suggestions you may have. A "Reader Comment Card" is located at the end of this of this manual for your use.

Before you begin, carefully read each of the procedures in this manual. Improper equipment handling may result in serious damage to the equipment.

This document contains information concerning the programming of the USP-1. It is intended for use by systems architects, programmers, technicians, and other users. Finally, the discussion retains a global and general view to be useful to a less technical user.

Please refer to the Themis USP-1 Hardware Manual for information concerning the USP-1 hardware.

1.2 In Case of Difficulties

Our Customer Support department is committed to providing the best product support in the industry. Customer support is available 8am - 5pm (PST), Monday through Friday via telephone, fax, e-mail or our World Wide Web site.

Themis Customer Support

Telephone: 510-252-0870
Fax: 510-490-5529
E-mail: support@themis.com
Web Site: http://www.themis.com

1.3 Contents

Chapters 1 and 2 provide introduction and installation instructions for the USP-1:

- Chapter 1, "Introduction," provides an introduction to the USP-1 including information on how to contact Themis and a chapter contents summary.
- Chapter 2, "Bus and Register Map," describes the Bus and address ranges of the USP-1.

Chapter 3 and 4 describe the VME Interface and VME Interface controller (SCV64) in detail:

- Chapter 3, "VME Interface," provides details on the VMEbus Interface and an overview of the VME Interface under Solaris 2.x.
- Chapter 4, "VME Interface ASIC: SCV64," contains a detailed description of the VMEbus controller, the SCV64.

Chapters 5 - 10 are concerned largely with the components common to the USP-1 and the UltraSPARC-1 Workstation:

- Chapter 5, "Programmer Visible State," describes the Physical Address Space Allocation, Memory Address Assignments, SBus System Controller Registers, and the I/O System.
- Chapter 6, "Interrupts," describes the initialization, servicing, and sources of interrupts. Chapter 6 also describes the Interrupt Concentrator, U2S Interrupt Handling, and UPA Slave Interrupt Bindings.
- Chapter 7, "SBus IOMMU," details the transformation of a virtual address to a physical address by the SBus IOMMU.
- Chapter 8, "Streaming Buffer," describe the streaming buffer implemented in the U2S.
- Chapter 9, "Error Handling," describes error detection and reporting and undetected errors.
- Chapter 10, "UPS Packet Handling," provides the USC Transaction Table.

2

Bus and Register Map

2.1 Introduction

This chapter contains memory maps for the USP-1 board and the VMEbus Interface.

2.2 UPA Port Address Ranges

This table specifies the physical address to UPA Port ID mapping in the System Controller (USC).

Table 2-1. **USC Address Map**

Address Range	Size	UPA Port Addressed	UPA Port ID
0x000.0000.0000 - 0x0FF.FFFF.FFFF	1TB	Main Memory	NA[a]
0x100.0000.0000 - 0x1BF.FFFF.FFFF	768 GB	reserved	NA
0x1C0.0000.0000 - 0x1C1.FFFF.FFFF	8 GB	processor 0	0 (0x00)
0x1C2.0000.0000 - 0x1C3.FFFF.FFFF	8 GB	reserved	1 (0x01)
0x1C4.0000.0000 - 0x1FB.FFFF.FFFF	224 GB	reserved	
0x1FC.0000.0000 - 0x1FD.FFFF.FFFF	8 GB	reserved	30 (0x1E)
0x1FE.0000.0000 - 0x1FF.FFFF.FFFF	8 GB	U2S (Systems I/O)	31 (0x1F)

a. The USC supports up to 1 GByte of memory. Addresses above 1GByte will wrap.

Note — Accesses to reserved regions will result in a time-out reply to the initiator.

2.3 Main Memory Address Assignments

Main memory on the USP-1 spans a 1 GByte region starting at physical address 0x000.0000.0000. The USP-1 can support either a single 128 MByte module or from one to four 256 MByte modules. Each module in the system has a unique address range based on the type and size of the module.

Table 2-2. **Main Memory Address Assignments**

Module Number	Type	Address Range (PA[30:0])
0	128 MBytes	0x0000.0000 - 0x07FF.FFFF
0	256 MBytes	0x0000.0000 - 0x0FFF.FFFF
1	128 MBytes	0x1000.0000 - 0x17FF.FFFF
1	256 MBytes	0x1000.0000 - 0x1FFF.FFFF
2	256 MBytes	0x2000.0000 - 0x2FFF.FFFF
3	256 MBytes	0x3000.0000 - 0x3FFF.FFFF

Note — Memory Modules #0 and #1 may be either 128 MBytes or 256 MBytes..

2.4 U2S Internal Registers

This table contains a list of registers internal to the U2S and ASIC.

Table 2-3. **U2S Internal Register Address Ranges**

Address	Register Name
0x1FE 0000 0000 - 0007	SYSIO_UPA_PortID
0x1FE 0000 0008 - 000F	SYSIO_UPA_Config
0x1FE 0000 0010 - 0017	SYSIO_Control/Status
0x1FE 0000 0020 - 0027	ECC_Control
0x1FE 0000 0030 - 0037	UE_AFSR (Uncorrectable Error Asynchronous Fault Status)
0x1FE 0000 0038 - 003F	UE_AFAR (Uncorrectable Error Asychronous Address)
0x1FE 0000 0040 - 0047	CE_AFSR (Correctable Error Asynchronous Status)
0x1FE 0000 0048 - 004F	CE_AFAR (Correctable Error Asynchronous Address)
0x1FE 0000 0100 - 0107	Performance Monitor Control
0x1FE 0000 0108 - 010F	Performance Counter
0x1FE 0000 2000 - 2007	SBus Control
0x1FE 0000 2010 - 2017	SBus AFSR

Table 2-3. **U2S Internal Register Address Ranges**

Address	Register Name
0x1FE 0000 2018 - 201F	SBus AFAR
0x1FE 0000 2020 - 2027	SBus Slot 0 Config
0x1FE 0000 2028 - 202F	SBus Slot 1 Config
0x1FE 0000 2030 - 2037	SBus Slot 2 Config
0x1FE 0000 2038 -203F	SBus Slot 3 Config
0x1FE 0000 2040 - 2047	SBus Slot 13 Config
0x1FE 0000 2048 - 204F	SBus Slot 14 Config
0x1FE 0000 2050 - 2057	SBus Slot 15 Config
0x1FE 0000 2400 - 2407	IOMMU Control
0x1FE 0000 2408 -240F	TSB Base Address
0x1FE 0000 2410 - 2417	IOMMU Flush
0x1FE 0000 2800 - 2807	Streaming Buffer Control
0x1FE 0000 2808 - 280F	Streaming Buffer Page Flush / Invalidate
0x1FE 0000 2810 - 2817	Streaming Buffer Flush Synchronization
0x1FE 0000 2C00 - 2C07	SBus Slot 0 Interrupt Mapping
0x1FE 0000 2C08 - 2C0F	SBus Slot 1 Interrupt Mapping
0x1FE 0000 2C10 - 2C17	SBus Slot 2 Interrupt Mapping
0x1FE 0000 2C18 - 2C1F	SBus Slot 3 Interrupt Mapping
0x1FE 0000 2C20 - 2C27	Interrupt Retry Timer
0x1FE 0000 3000 - 3007	SCSI Interrupt Mapping
0x1FE 0000 3008 - 300F	Ethernet Interrupt Mapping
0x1FE 0000 3010 - 3017	Parallel Port Interrupt Mapping
0x1FE 0000 3018 - 301F	Audio Interrupt Mapping
0x1FE 0000 3020 - 3027	Power Fail Interrupt Mapping
0x1FE 0000 3028 - 302F	Keyboard / Mouse / Serial Interrupt Mapping
0x1FE 0000 3030 - 3037	Floppy Interrupt Mapping
0x1FE 0000 3038 - 303F	Thermal Warning Interrupt Mapping
0x1FE 0000 3060 - 3067	Timer 0 Interrupt Mapping
0x1FE 0000 3068 - 306F	Timer 1 Interrupt Mapping
0x1FE 0000 3070 - 3077	UE Interrupt Mapping
0x1FE 0000 3078 - 307F	CE Interrupt Mapping
0x1FE 0000 3080 - 3087	SBus Error Interrupt Mapping
0x1FE 0000 3088 - 308F	Power Management Wakeup Interrupt Mapping
0x1FE 0000 3090 - 3097	UPA Expansion (Graphics) Interrupt Mapping
0x1FE 0000 3098 - 309F	Reserved Interrupt Mapping

Table 2-3. **U2S Internal Register Address Ranges**

Address	Register Name
0x1FE 0000 3408 - 3437	SBus Slot 0 Clear Interrupt
0x1FE 0000 3448 - 3478	SBus Slot 1 Clear Interrupt
0x1FE 0000 3488 - 34B7	SBus Slot 2 Clear Interrupt
0x1FE 0000 34C8 - 34F8	SBus Slot 3 Clear Interrupt
0x1FE 0000 3800 - 3807	SCSI Clear Interrupt
0x1FE 0000 3808 - 380F	Ethernet Clear Interrupt
0x1FE 0000 3810 - 3817	Parallel Port Clear Interrupt
0x1FE 0000 3818 - 381F	Audio Clear Interrupt
0x1FE 0000 3820 - 3827	Power Fail Clear Interrupt
0x1FE 0000 3828 - 382F	Keyboard / Mouse / Serial Clear Interrupt
0x1FE 0000 3830 - 3837	Floppy Clear Interrupt
0x1FE 0000 3838 - 383F	Thermal Warning Clear Interrupt
0x1FE 0000 3860 - 3867	Timer 0 Clear Interrupt
0x1FE 0000 3868 - 386F	Timer 1 Clear Interrupt
0x1FE 0000 3870 - 3877	UE Clear Interrupt
0x1FE 0000 3878 - 387F	CE Clear Interrupt
0x1FE 0000 3880 - 387F	SBus Asynchronous Error Clear Interrupt
0x1FE 0000 3888 - 388F	Power Management Wakeup Clear Interrupt
0x1FE 0000 3C00 - 3C07	Timer / Counter 0 Count
0x1FE 0000 3C08 - 3C0F	Timer / Counter 0 Limit
0x1FE 0000 3C10 - 3C17	Timer / Counter 1 Count
0x1FE 0000 3C18 - 3C1F	Timer / Counter 1 Limit
0x1FE 0000 4400 - 4407	SBus VA Diagnostic
0x1FE 0000 4408 - 440F	TLB Tag Compare Diag
0x1FE 0000 4500 - 457F	IOMMU LRU Queue Diagnostic
0x1FE 0000 4580 - 45FF	TLB Tag Diagnostic
0x1FE 0000 4600 - 46FF	TLB Data RAM Diag
0x1FE 0000 4800 - 4807	SBus Interrupt State Diag
0x1FE 0000 4808 - 480F	OBIO and Misc. Interrupt State Diag
0x1FE 0000 5000 - 53FF	Streaming Buffer Data RAM Diagnostic
0x1FE 0000 5400 - 57FF	Streaming Buffer Error Status Diagnostic
0x1FE 0000 5800 - 587F	Streaming Buffer Page Tag Diagnostic
0x1FE 0000 5900 - 597F	Streaming Buffer Line Tag Diagnostic

2.5 U2S to SBus Address Mapping

This table provides the physical address to SBus mapping sizes, and allocations of the SBus slots.

Table 2-4. **U2S (SYSTEM I/O) SBus Address Assignments**

Address Range in UPA Address <40:0>	Size	SBus PORT Addressed	Allocation
0x1FF.0000.0000 - 0x1FF.0FFF.FFFF	256 MByte	Slot 0	SBus or VMEbus
0x1FF.1000.0000 - 0x1FF.FFFF.FFFF	256 MByte	Slot 1	SBus or VMEbus
0x1FF.2000.0000 - 0x1FF.2FFF.FFFF	256 MByte	Slot 2	VMEbus
0x1FF.3000.0000 - 0x1FF.3FFF.FFFF	256 MByte	Slot 3	MACIO #2 and VMEbus
0x1FF.4000.0000 - 0x1FF.CFFF.FFFF	2.256 GByte	Reserved	--
0x1FF.D000.0000 - 0x1FF.DFFF.FFFF	256 MByte	Slot 13	APC and VMEbus
0x1FF.E000.0000 - 0x1FF.EFFF.FFFF	256 MByte	Slot 14	MACIO #1and VMEbus
0x1FF.F000.0000 - 0x1FF.FFFF.FFFF	256 MByte	Slot 15	SLAVIO, VMEbus, VSIC, VMMU, IACK

2.6 Register and Device Addresses

This series of tables provides the address, name, and reference for each device and register handled by the SBus.

2.6.1 Slot 3: MACIO #2 and VMEbus

2.6.1.1 Slot 3 Address Map

Table 2-5. **Slot 3 (Auxiliary MACIO and VMEbus) mapping**

UPA Address	Device
0x1FF.3000.0000 - 0x1FF.38FF.FFFF	VMEbus
0x1FF.3900.0000 - 0x1FF.39FF.FFFF	MACIO #2 Internal Registers
0x1FF.3A00.0000 - 0x1FF.3FFF.FFFF	VMEbus

2.6.1.2 MACIO #2 Address Map

- **Address Range: 0x1FF.3000.0000 - 0x1FF.3FFF.FFFF**
- **Size: 256 MByte**

- **Slot 3**

Table 2-6. **Auxiliary MACIO Register Address Range**

Address Range in UPA Address	Register Name
0x1FF.3900.0000 - 0x1FF.3000.0003	**DMA2 Internal ID Register**
0x1FF.3920.0000 - 0x1FF.3920.000F	**DMA2 ESP Registers**
0x1FF.3920.0000 - 0x1FF.3920.0003	Control / Status Register
0x1FF.3920.0004 - 0x1FF.3920.0007	Address Register
0x1FF.3920.0008 - 0x1FF.3920.000B	Byte Count Register
0x1FF.3920.000c - 0x1FF.3920.000F	Test Control / Status Register
0x1FF.3920.0010 - 0x1FF.3920.001F	**DMA2 Ethernet Registers**
0x1FF.3920.0010 - 0x1FF.3920.0013	Control / Status Register
0x1FF.3920.0014 - 0x1FF.3920.0017	Test Control / Status Register
0x1FF.3920.0018 - 0x1FF.3920.001B	Cache Valid Bits
0x1FF.3920.001C - 0x1FF.3920.001F	Base Address Register
0x1FF.3940.0000 - 0x1FF.3940.003F	**SCSI Controller Registers**
0x1FF.3940.0000	Transfer Count Low (7:0)
0x1FF.3940.0004	Transfer Count Middle (15:8)
0x1FF.3940.0008	FIFO Data
0x1FF.3940.000C	Command
0x1FF.3940.0010	Status (Read Only)
0x1FF.3940.0010	Select-Reselect Bus ID (Write Only)
0x1FF.3940.0014	Interrupt (Read Only)
0x1FF.3940.0014	Select-Reselect Time-out (Write Only)
0x1FF.3940.0018	Sequence Step (Read Only)
0x1FF.3940.0018	Synchronous Transfer Period (Write Only)
0x1FF.3940.001C	FIFO Flags (Read Only)
0x1FF.3940.001C	Synchronous Offset (Write Only)
0x1FF.3940.0020	Configuration #1
0x1FF.3940.0024	Clock Conversion Factor (Write Only)
0x1FF.3940.0028	Test (Chip Test Use Only)
0x1FF.3940.002C	Configuration #2
0x1FF.3940.0030	Configuration #3
0x1FF.3940.0038	Transfer High Count #3 (23:16)
0x1FF.3960.0000 - 0x1FF.3960.0003	**Ethernet Controller Registers**
0x1FF.3960.0000	Register Data Port
0x1FF.3960.0002	Register Address Port

2.6.2 Slot 13: APC and VMEbus

2.6.2.1 Slot 13 Address Map

Table 2-7. **Slot 13 Address map**

Address Range in UPA Address <40:0>	Function
0x1FF.D000.0000 - 0x1FF.D9FF.FFFF	VMEbus
0x1FF.DA00.0000 - 0x1FF.DAFF.FFFF	APC Registers
0x1FF.DB00.0000 - 0x1FF.DBFF.FFFF	VMEbus
0x1FF.DC00.0000 - 0x1FF.DCFF.FFFF	APC Internal Registers
0x1FF.DD00.0000 - 0x1FF.DDFF.FFFF	VMEbus
0x1FF.DE00.0000 - 0x1FF.DEFF.FFFF	APC Internal Registers
0x1FF.DF00.0000 - 0x1FF.DFFF.FFFF	VMEbus

2.6.2.2 APC Address Map

- **Address Range: 0x1FF.D000.0000 - 0x1FF.DFFF.FFFF**
- **Size: 256 MByte**
- **Slot 13**

Table 2-8. **APC Register Addresses**

Addresses	Register Name
0x1FF.DA00.0000	Idle Register
0x1FF.DA00.0020	Fan Control Register
0x1FF.DA00.0024	Convenience Outlet Shutoff Register
0x1FF.DA00.0030	Generic Bits Port

Table 2-9. **Audio DMA Registers**

Addresses	Register Name
0x1FF.DC00.0000	Codec Address Port
0x1FF.DC00.0004	Codec Data Port
0x1FF.DC00.0008	Codec Status Port
0x1FF.DC00.000C	Codec PIO Data Port
0x1FF.DC00.0010 - 0x1FF.DC00.0013	Control / Status (CSR)
0x1FF.DC00.0014	Reserved
0x1FF.DC00.0020 - 0x1FF.DC00.0023	Capture Virtual Address (CVA)
0x1FF.DC00.0024 - 0x1FF.DC00.0025	Capture Word Count (CC)
0x1FF.DC00.0028 - 0x1FF.DC00.002B	Capture Next VA (CNVA)
0x1FF.DC00.002C - 0x1FF.DC00.002D	Capture Next Word Count (CNC)

Table 2-9. **Audio DMA Registers**

Addresses	Register Name
0x1FF.DC00.0030 - 0x1FF.DC00.0033	Playback Virtual Address (PVA)
0x1FF.DC00.0034 - 0x1FF.DC00.0035	Playback Word Count (PC)
0x1FF.DC00.0038 - 0x1FF.DC00.003B	Playback Next VA (PNVA)
0x1FF.DC00.003C - 0x1FF.DC00.003D	Playback Next Word Count (PNC)
0x1FF.DE00.0000	AFX Register

2.6.3 Slot 14: MACIO #1 and VMEbus

2.6.3.1 Slot 14 Address Map

Table 2-10. **Slot 14 Address Map**

Address Range, UPA Address <40:0>	Function
0x1FF.E000.0000 - 0x1FF.E7FFF.FFFF	VMEbus
0x1FF.E800.0000 - 0x1FF.E8FFF.FFFF	MACIO #1 Internal Registers
0x1FF.E900.0000 - 0x1FF.E9FFF.FFFF	VMEbus
0x1FF.EC00.0000 - 0x1FF.ECFFF.FFFF	MACIO #1 Internal Registers
0x1FF.ED00.0000 - 0x1FF.EDFFF.FFFF	VMEbus

2.6.3.2 MACIO Address Map

- **Address Range: 0x1FF.E000.0000 - 0x1FF.EFFF.FFFF**
- **Size: 256 MByte**
- **Slot 14**

Table 2-11. **MACIO #1 Internal Registers**

Address Ranges	Register Name
0x1FF.E800.0000	**DMA2 Internal ID Register**
0x1FF.E840.0000 - 0x1FF.E840.000F	**DMA2 ESP Registers**
0x1FF.E840.0000	Control / Status Register
0x1FF.E840.0004	Address Register
0x1FF.E840.0008	Byte Count Register
0x1FF.E840.000C	Test Control / Status Register
0x1FF.E840.0010 - 0x1FF.E840.001F	**DMA2 Ethernet Registers**
0x1FF.E840.0010	Control / Status Register
0x1FF.E840.0014	Test Control / Status Register
0x1FF.E840.0018	Cache Valid Bits
0x1FF.E840.001C	Base Address Register

Table 2-11. **MACIO #1 Internal Registers**

Address Ranges	Register Name
0x1FF.E880.0000 - 0x1FF.E880.003F	**SCSI Controller Registers**
0x1FF.E880.0000	Transfer Count Low (7:0)
0x1FF.E880.0004	Transfer Count Middle (15:8)
0x1FF.E880.0008	FIFO Data
0x1FF.E880.000C	Command
0x1FF.E880.0010	Status (Read Only)
0x1FF.E880.0010	Select - Reselect Bus ID (Write Only)
0x1FF.E880.0014	Interrupt (Read Only)
0x1FF.E880.0014	Select - Reselect Time-out (Write Only)
0x1FF.E880.0018	Sequence Step (Read Only)
0x1FF.E880.0018	Synchronous Transfer Period (Write Only)
0x1FF.E880.001C	FIFO Flags (Read Only)
0x1FF.E880.001C	Synchronous Offset (Write Only)
0x1FF.E880.0020	Configuration #1
0x1FF.E880.0024	Clock Conversion Factor (Write Only)
0x1FF.E880.0028	Test (Chip Test Use Only)
0x1FF.E880.002C	Configuration #2
0x1FF.E880.0030	Configuration #3
0x1FF.E880.0038	Transfer Count High (23:16)
0x1FF.E8C0.0000 - 0x1FF.E8C0.003	**Ethernet Controller Registers**
0x1FF.E8C0.0000	Register Data Port
0x1FF.E8C0.0002	Register Address Port
0x1 FF.EC80.0000 - 0x1FF.3E80.001F	**DMA2 Parallel Port Registers**
0x1FF.EC80.0000	Control / Status Register
0x1FF.EC80.0004	Address Register
0x1FF.EC80.0008	Byte Count Register
0x1FF.EC80.000C	Test Control / Status Register
0x1FF.EC80.0010	Hardware Configuration Register
0x1FF.EC80.0012	Operation Configuration Register
0x1FF.EC80.0014	Parallel Data Register
0x1FF.EC80.0015	Transfer Control Register
0x1FF.EC80.0016	Output Register
0x1FF.EC80.0017	Input Register
0x1FF.EC80.0018	Interrupt Control Register

2.6.4 Slot 15: SLAVIO and VMEbus

2.6.4.1 Slot 15 Address Map

Table 2-12. **Slot 15 Address Map**

Address Range, UPA Address <40:0>	Function
0x1FF.F000.0000 - 0x1FF.F0FF.FFFF	Boot Device
0x1FF.F100.0000 - 0x1FF.F1FF.FFFF	SLAVIO Internal and External Registers
0x1FF.F200.0000 - 0x1FF.FBFF.FFFF	VMEbus
0x1FF.FC00.0000 - 0x1FF.FCFF.FFFF	VSIC Registers
0x1FF.FD00.0000 - 0x1FF.FDFF.FFFF	SCV64 Internal Registers
0x1FF.FE00.0000 - 0x1FF.FEFF.FFFF	VMMU
0x1FF.FF00.0000 - 0x1FF.FFFF.FFFF	IACK Registers

2.6.4.2 External and VSIC Register Address Map

Table 2-13. **External Register Address Map**

Address Range in UPA Address <40:0>	Register Name
0x1FF.F1C0.0000	Board Status Register
0x1FF.F1C0.0004	LED Register
0x1FF.F1C0.0008	Switch Register
0x1FF.F1C0.000C	Reset DS1620 Thermostat
0x1FF.F1C0.0010	DS1620 Serial I/O
0x1FF.F1C0.0020	Command Register
0x1FF.FC00.0000	VSIC ID Register

2.6.4.3 SLAVIO Address Map

- **Address Range: 0x1FF.F000.0000 - 0x1FF.FFFF.FFFF**
- **Size: 256 MByte**
- **Slot 15**

Table 2-14. **SLAVIO Address Map**

Address Ranges	Device Name
0x1FF.F000.0000 - 0x1FF.F007.FFFF	**Boot FLASH (Read Only)**
0x1FF.F100.0000 - 0x1FF.F1FF.FFFF	**Keyboard, Mouse, and Serial Ports**
0x1FF.F100.0000	Mouse Control Port
0x1FF.F100.0002	Mouse Data Port
0x1FF.F100.0004	Keyboard Control Port

Table 2-14. **SLAVIO Address Map**

Address Ranges	Device Name
0x1FF.F100.0006	Keyboard Data Port
0x1FF.F110.0000	TTYB Control Port
0x1FF.F110.0002	TTYB Data Port
0x1FF.F110.0004	TTYA Control Port
0x1FF.F110.0006	TTYA Data Port
0x1FF.F120.0000 - 0x1FF.F12F.FFFF	**TOD/NVRAM**
0x1FF.F130.0000 - 0x1FF.F13F.3FFF	**General Purpose (Generic Port)**
0x1FF.F130.0000	SC_Address Register Number [a]
0x1FF.F130.0004	SC_Data Register Number [b]
0x1FF.F130.4000	Frequency Margining Serial Load
0x1FF.F130.4001	Frequency Margining Serial Load and Reset
0x1FF.F130.4002	Frequency Margining Serial Data In
0x1FF.F138.0000 - 0x1FF.F13F.FFFF	Boot Flash (Write Only)
0x1FF.F140.0000 - 0x1FF.F14F.FFFF	**Floppy Controller**
0x1FF.F140.0000	Digital Output Register
0x1FF.F140.0004	Main Status Register (Read Only)
0x1FF.F140.0004	Data Rate Select Register
0x1FF.F140.0005	FIFO
0x1FF.F140.0006	Reserved
0x1FF.F140.0007	Digital Input Register (Read Only)
0x1FF.F140.0007	Configuration Control Register (Write Only)
0x1FF.F150.0000 - 0x1FF.F17F.FFFF	**Reserved**
0x1FF.F180.0000	**89C105 Configuration Register**
0x1FF.F190.0000 - 0x1FF.F19F.FFFF	**Auxiliary I/O Registers**
0x1FF.F190.0000	Auxiliary 1 Register (Miscellaneous System Functions)
0x1FF.F191.0000	Auxiliary 2 Register (Software Power-down Control)
0x1FF.F1A0.0000	**Diagnostic Message Register**
0x1FF.F1B0.0000	**Modem Register**
0x1FF.F1C0.0000 - 0x1FF.F1CF.FFFF	**Reserved**
0x1FF.F1D0.0000 - 0x1FF.F1DF.FFFF	**Counters / Timers**
0x1FF.F1D0.0000	Processor Counter Limit Register or User Timer MSW
0x1FF.F1D0.0004	Processor Counter Register or User Time LSW
0x1FF.F1D0.0008	Processor Counter Limit Register (non-resetting port)
0x1FF.F1D0.000C	Processor Counter User TImer Start / Stop Register
0x1FF.F1D1.0000	System Limit Register (Level 10 Interrupt)

Table 2-14. **SLAVIO Address Map**

Address Ranges	Device Name
0x1FF.F1D1.0004	System Counter Register
0x1FF.F1D1.0008	System Limit Register (non-resetting port)
0x1FF.F1D1.000C	Reserved
0x1FF.F1D1.0010	Timer Configuration Register
0x1FF.F1E0.0000 - 0x1FF.F1EF.FFFF	**Interrupt Controller**
0x1FF.F1E0.0000	Processor Interrupt Pending Register
0x1FF.F1E0.0004	Processor Clear-Pending Pseudo-Register
0x1FF.F1E0.0008	Processor Set-Soft_Interrupt Pseudo-Register
0x1FF.F1E1.0000	System Interrupt Pending Register
0x1FF.F1E1.0004	Interrupt Target Mask Register
0x1FF.F1E1.0008	Interrupt Target Mask Clear Pseudo-Register
0x1FF.F1E1.000C	Interrupt Target Mask Set Pseudo-Register
0x1FF.F1E1.0010	Interrupt Target Register (Reads as 0, write has no effect)
0x1FF.F1F0.0000 - 0x1FF.F1F0.0003	**System Control / Status Register**

a. Physically Located in the USC.
b. Physically Located in the USC.

2.7 VMEbus Mapping

This section contains the address mapping tables for the VME MMU and the SCV64 Controller Register Map.

2.7.1 VME MMU Address Translation Table

The following table gives the specifier for the base address for each UPA address.

Table 2-15. **VME MMU Addresses**

UPA Address Range <40:0>	Specifier for Base Address
0x1FF.FE00.0BE0	0x1FF.F000.0000
0x1FF.FE00.0BE4	0x1FF.F200.0000
0x1FF.FE00.0BE8	0x1FF.F400.0000
0x1FF.FE00.0BEC	0x1FF.F600.0000
0x1FF.FE00.0BF0	0x1FF.F800.0000
0x1FF.FE00.0BF4	0x1FF.FA00.0000
0x1FF.FE00.0BF8	0x1FF.FC00.0000

Table 2-15. **VME MMU Addresses**

UPA Address Range <40:0>	Specifier for Base Address
0x1FF.FE00.0BFC	0x1FF.FE00.0000
0x1FF.FE00.0FE0	0x1FF.F100.0000
0x1FF.FE00.0FE4	0x1FF.F300.0000
0x1FF.FE00.0FE8	0x1FF.F500.0000
0x1FF.FE00.0FEC	0x1FF.F700.0000
0x1FF.FE00.0FF0	0x1FF.F900.0000
0x1FF.FE00.0FF4	0x1FF.FB00.0000
0x1FF.FE00.0FF8	0x1FF.FD00.0000
0x1FF.FE00.0FFC	0x1FF.FF00.0000
0x1FF.FE00.19E0	0x1FF.D000.0000
0x1FF.FE00.19E4	0x1FF.D200.0000
0x1FF.FE00.19E8	0x1FF.D400.0000
0x1FF.FE00.19EC	0x1FF.D600.0000
0x1FF.FE00.19F0	0x1FF.D800.0000
0x1FF.FE00.19F4	0x1FF.DA00.0000
0x1FF.FE00.19F8	0x1FF.DC00.0000
0x1FF.FE00.19FC	0x1FF.DE00.0000
0x1FF.FE00.1AE0	0x1FF.3000.0000
0x1FF.FE00.1AE4	0x1FF.3200.0000
0x1FF.FE00.1AE8	0x1FF.3400.0000
0x1FF.FE00.1AEC	0x1FF.3600.0000
0x1FF.FE00.1AF0	0x1FF.3800.0000
0x1FF.FE00.1AF4	0x1FF.3A00.0000
0x1FF.FE00.1AF8	0x1FF.3C00.0000
0x1FF.FE00.1AFC	0x1FF.3E00.0000
0x1FF.FE00.1B60	0x1FF.2000.0000
0x1FF.FE00.1B64	0x1FF.2200.0000
0x1FF.FE00.1B68	0x1FF.2400.0000
0x1FF.FE00.1B6C	0x1FF.2600.0000
0x1FF.FE00.1B70	0x1FF.2800.0000
0x1FF.FE00.1B74	0x1FF.2A00.0000
0x1FF.FE00.1B78	0x1FF.2C00.0000
0x1FF.FE00.1B7C	0x1FF.2E00.0000
0x1FF.FE00.1BA0	0x1FF.1000.0000
0x1FF.FE00.1BA4	0x1FF.1200.0000

Table 2-15. **VME MMU Addresses**

UPA Address Range <40:0>	Specifier for Base Address
0x1FF.FE00.1BA8	0x1FF.1400.0000
0x1FF.FE00.1BAC	0x1FF.1600.0000
0x1FF.FE00.1BB0	0x1FF.1800.0000
0x1FF.FE00.1BB4	0x1FF.1A00.0000
0x1FF.FE00.1BB8	0x1FF.1C00.0000
0x1FF.FE00.1BBC	0x1FF.1E00.0000
0x1FF.FE00.1BC0	0x1FF.0000.0000
0x1FF.FE00.1BC4	0x1FF.0200.0000
0x1FF.FE00.1BC8	0x1FF.0400.0000
0x1FF.FE00.1BCC	0x1FF.0600.0000
0x1FF.FE00.1BD0	0x1FF.0800.0000
0x1FF.FE00.1BD4	0x1FF.0A00.0000
0x1FF.FE00.1BD8	0x1FF.0C00.0000
0x1FF.FE00.1BDC	0x1FF.0E00.0000
0x1FF.FE00.1BE0	0x1FF.E000.0000
0x1FF.FE00.1BE4	0x1FF.E200.0000
0x1FF.FE00.1BE8	0x1FF.E400.0000
0x1FF.FE00.1BEC	0x1FF.E600.0000
0x1FF.FE00.1BF0	0x1FF.E800.0000
0x1FF.FE00.1BF4	0x1FF.EA00.0000
0x1FF.FE00.1BF8	0x1FF.EC00.0000
0x1FF.FE00.1BFC	0x1FF.EE00.0000
0x1FF.FE00.1DE0	0x1FF.D100.0000
0x1FF.FE00.1DE4	0x1FF.D300.0000
0x1FF.FE00.1DE8	0x1FF.D500.0000
0x1FF.FE00.1DEC	0x1FF.D700.0000
0x1FF.FE00.1DF0	0x1FF.D900.0000
0x1FF.FE00.1DF4	0x1FF.DB00.0000
0x1FF.FE00.1DF8	0x1FF.DD00.0000
0x1FF.FE00.1DFC	0x1FF.DF00.0000
0x1FF.FE00.1EE0	0x1FF.3100.0000
0x1FF.FE00.1EE4	0x1FF.3300.0000
0x1FF.FE00.1EE8	0x1FF.3500.0000
0x1FF.FE00.1EEC	0x1FF.3700.0000
0x1FF.FE00.1EF0	0x1FF.3900.0000

Table 2-15. **VME MMU Addresses**

UPA Address Range <40:0>	Specifier for Base Address
0x1FF.FE00.1EF4	0x1FF.3B00.0000
0x1FF.FE00.1EF8	0x1FF.3D00.0000
0x1FF.FE00.1EFC	0x1FF.3F00.0000
0x1FF.FE00.1F60	0x1FF.2100.0000
0x1FF.FE00.1F64	0x1FF.2300.0000
0x1FF.FE00.1F68	0x1FF.2500.0000
0x1FF.FE00.1F6C	0x1FF.2700.0000
0x1FF.FE00.1F70	0x1FF.2900.0000
0x1FF.FE00.1F74	0x1FF.2B00.0000
0x1FF.FE00.1F78	0x1FF.2D00.0000
0x1FF.FE00.1F7C	0x1FF.2F00.0000
0x1FF.FE00.1FA0	0x1FF.1100.0000
0x1FF.FE00.1FA4	0x1FF.1300.0000
0x1FF.FE00.1FA8	0x1FF.1500.0000
0x1FF.FE00.1FAC	0x1FF.1700.0000
0x1FF.FE00.1FB0	0x1FF.1900.0000
0x1FF.FE00.1FB4	0x1FF.1B00.0000
0x1FF.FE00.1FB8	0x1FF.1D00.0000
0x1FF.FE00.1FBC	0x1FF.1F00.0000
0x1FF.FE00.1FC0	0x1FF.0100.0000
0x1FF.FE00.1FC4	0x1FF.0300.0000
0x1FF.FE00.1FC8	0x1FF.0500.0000
0x1FF.FE00.1FCC	0x1FF.0700.0000
0x1FF.FE00.1FD0	0x1FF.0900.0000
0x1FF.FE00.1FD4	0x1FF.0B00.0000
0x1FF.FE00.1FD8	0x1FF.0D00.0000
0x1FF.FE00.1FDC	0x1FF.0F00.0000
0x1FF.FE00.1FE0	0x1FF.E100.0000
0x1FF.FE00.1FE4	0x1FF.E300.0000
0x1FF.FE00.1FE8	0x1FF.E500.0000
0x1FF.FE00.1FEC	0x1FF.E700.0000
0x1FF.FE00.1FF0	0x1FF.E900.0000
0x1FF.FE00.1FF4	0x1FF.EB00.0000
0x1FF.FE00.1FF8	0x1FF.ED00.0000
0x1FF.FE00.1FFC	0x1FF.EF00.0000

Table 2-16. **IACK Registers**

UPA Address <40:0>	Register Name
0x1FF.FF00.0000	VMMU Enable Register
0x1FF.FF00.0004	VMEIRQ1 Interrupt Acknowledge Register
0x1FF.FF00.0008	VMEIRQ2 Interrupt Acknowledge Register
0x1FF.FF00.000C	VMEIRQ3 Interrupt Acknowledge Register
0x1FF.FF00.0010	VMEIRQ4 Interrupt Acknowledge Register
0x1FF.FF00.0014	VMEIRQ5 Interrupt Acknowledge Register
0x1FF.FF00.0018	VMEIRQ6 Interrupt Acknowledge Register
0x1FF.FF00.001C	VMEIRQ7 Interrupt Acknowledge Register
0x1FF.FF00.0400	VSIC/SCV64 Interrupt Status Register
0x1FF.FF00.0404	LIRQ1 Interrupt Acknowledge Register
0x1FF.FF00.0408	LIRQ2 Interrupt Acknowledge Register
0x1FF.FF00.040C	LIRQ3 Interrupt Acknowledge Register
0x1FF.FF00.0410	LIRQ4 Interrupt Acknowledge Register
0x1FF.FF00.0414	LIRQ5 Interrupt Acknowledge Register
0x1FF.FF00.0418	LIRQ6 Interrupt Acknowledge Register
0x1FF.FF00.051C	LIRQ7 Interrupt Acknowledge Register

2.7.2 SCV64 Register Addresses

The following table contains the name and address of the SCV64 registers. For more information concerning the SCV64 Registers refer to Chapter 4, VMEbus Interface ASIC: SCV64.

Table 2-17. **SCV64 Register Addresses**

Address	Register	Name
0x1FF FD00 0000	DMA Local Address	DMALAR
0x1FF FD00 0004	DMA VMEbus Address	DMAVAR
0x1FF FD00 0008	DMA Transfer Count	DMATC
0x1FF FD00 000C	Control and Status	DCSR
0x1FF FD00 0010	VMEbus Slave Base Address	VMEBAR
0x1FF FD00 0014	Rx FIFO Data	RXDATA
0x1FF FD00 0018	Rx FIFO Address Register	RXADDR
0x1FF FD00 001C	Rx FIFO Control Register	RXCTL
0x1FF FD00 0020	VMEbus / VSB Bus Select	BUSSEL
0x1FF FD00 0024	VMEbus Interrupter Vector	IVECT
0x1FF FD00 0028	Access Protect Boundary	APBR

Table 2-17. **SCV64 Register Addresses**

Address	Register	Name
0x1FF FD00 002C	Tx FIFO Data Output Latch	TXDATA
0x1FF FD00 0030	Tx FIFO Address Output Latch	TXADDR
0x1FF FD00 0034	Tx FIFO AM Code and Control Bit Latch	TXCTL
0x1FF FD00 0038	Location Monitor FIFO Read Port	LMFIFO
0x1FF FD00 003C	SCV64 Mode Control	MODE
0x1FF FD00 0040	Slave A64 Base Address	SA64BAR
0x1FF FD00 0044	Master A64 Base Address	MA64BAR
0x1FF FD00 0048	Local Address Generator	LAG
0x1FF FD00 004C	DMA VMEbus Transfer Count	DMAVTC
0x1FF FD00 0050 - 007F	Reserved	
0x1FF FD00 0080	Status Register 0	STAT0
0x1FF FD00 0084	Status Register 1	STAT1
0x1FF FD00 0088	General Control Register	GENCTL
0x1FF FD00 008C	VMEbus Interrupter Requester	VINT
0x1FF FD00 0090	VMEbus Requester Register	VREQ
0x1FF FD00 0094	VMEbus Arbiter Register	VARB
0x1FF FD00 0098	ID Register	ID
0x1FF FD00 009C	Control and Status Register	CTL2
0x1FF FD00 00A0	Level 7 Interrupt Status Register	7IS
0x1FF FD00 00A4	Local Interrupt Status Register	LIS
0x1FF FD00 00A8	Level 7 Interrupt Enable Register	7IE
0x1FF FD00 00AC	Local Interrupt Enable Register	LIE
0x1FF FD00 00B0	VMEbus Interrupt Enable Register	VIE
0x1FF FD00 00B4	Local Interrupts 1 and 0 Control Register	IC10
0x1FF FD00 00B8	Local Interrupts 3 and 2 Control Register	IC32
0x1FF FD00 00BC	Local Interrupts 5 and 4 Control Register	IC54
0x1FF FD00 00C0	Miscellaneous Control Register	MISC
0x1FF FD00 00C4	Delay Line Control Register	DLCT
0x1FF FD00 00C8	Delay Line Status Register 1	DLST1
0x1FF FD00 00CC	Delay Line Status Register 2	DLST2
0x1FF FD00 00D0	Delay Line Status Register 3	DLST3
0x1FF FD00 00D4	Mailbox Register 0	MBOX0
0x1FF FD00 00D8	Mailbox Register 1	MBOX1
0x1FF FD00 00DC	Mailbox Register 2	MBOX2

Table 2-17. **SCV64 Register Addresses**

Address	Register	Name
0x1FF FD00 00E0	Mailbox Register 3	MBOX3
0x1FF FD00 00E4 - 01FC	Reserved	

3.1 Introduction

This guide is intended to provide helpful information and tips to systems programmers who use the USP-1 single board computer. The next section is intended to address the programmers who wish to program the USP-1 for the VMEbus without the aid of an operating system or real-time kernel. Further sections are intended to address the programmers who wish to access the USP-1 from any of the supported operating system environments.

Detailed information concerning on-board device interfaces, memory maps and device accesses are contained in the separate chapters and should be consulted while reading this guide.

The USP-1 processor board implements a Sun4u compatible workstation on a double-width 6U VMEbus board. A full VME master/slave interface is included, using the Tundra SCV64 VMEbus controller paired with the Themis KSIC SBus to SCV64 interface. The interface is referred to as the VSIC: VMEbus to SBus Interface Controller.

This guide is primarily concerned with the implementation and programming of Themis USP-1 specific features that are not common to the Sun SBus Reference Platform System. Information on the SUN SBus Reference Platform System may be found in Chapters 4 - 10.

3.2 USP-1 VMEbus Interface

3.2.1 Primary Features

The VSIC interface is composed of the Tundra SCV64 VMEbus interface controller chip and the bus protocol converter logic (KSIC) responsible for bridging the SCV64 local bus to the SBus. For details on the SCV64 VMEbus controller please refer to Chapter 4, VMEbus Interface Implementation, or the Tundra SCV64 User's Manual. (TundraSemiconductor Corp., 603 March Road, Kanata ON Canada K2K 2M5 or at http://www.tundra.com).

The SCV64 offers the following major features:

- VMEbus System Controller functions

- VMEbus Master Interface capability
- VMEbus Slave Interface capability
- VMEbus Interrupter
- VMEbus Interrupt handler
- Build-in DMA engine.

Other attractive features:

- Automatic VMEbus SYSCON identification
- Location Monitor
- VME slave access protection
- Auto-ID VME slot identification
- Bus isolation
- SYSRESET* Generation.

3.2.2 System Controller

The USP-1 features a slot one system controller including SYSCLK, SYSRESET, bus time-out, and a four level arbiter. System Controller capabilities are enabled by installing jumper JP1201 1-2 (see Chapter 4 in the Hardware Manual, Configurations and Options). By installing jumper, JP0401, the VSIC can generate and receive VMEbus SYSRESET*.

Both single arbitration or round-robin arbiters are supported.

Slot one System Controller features include:

- Programmable VMEbus arbitration schemes:
  - Full priority on all four request levels
  - Round-robin priority on all four request levels
  - Variant of both.
- VMEbus arbitration timer (16µs):
  - 16µs Maximum bus grant propagation time through the daisy-chain.
- Programmable VMEbus request mode:
  - Unconditional (default setting)
  - Fair mode.
- Programmable VMEbus ownership period:
  - No time-out
  - 2 µs
  - 4 µs
  - 8 µs (default setting).
- Programmable VMEbus data transfer time-out period:
  - 16 µs
  - 32 µs

  - 64 µs (default setting)
  - No time-out.
- Programmable VMEbus release mode
  - Release when done (RWD)
  - Release on request (ROR) default setting
  - Release on Bus Clear.
- System clock driver (16 MHz).
- IACK daisy chain driver.

3.2.3 VMEbus Master Interface

The master interface of the USP-1 supports 8, 16, and 32-bit data transfer cycles in A32, A24 and A16 addressing modes. In DMA mode, A64 addressing capabilities are also supported. A16/D32 transfer is not supported.

The VMEbus is made accessible to the UltraSPARC-I processor by mapping the unused SBus addresses through the VME MMU translation table map. This MMU allows the user to selectively map each unused 16 MByte segment of an SBus slot to the VMEbus.

Themis has implemented logic to translate the SBus transactions of the USP-I system I/O to the 68K-oriented local bus of the SCV64 VME controller. This interface, named KSIC is described in detail.

The KSIC interface and the SCV64 controller operate in conjunction with the VME MMU map table to provide flexible mapping of SBus address ranges to non-contiguous VMEbus addresses. As described earlier, each SBus slot is divided into 16 MByte segments. The VMU MMU must be programmed prior to initiating. The user is responsible for configuring the VME MMU to match their own unique system requirements.

When the CPU initiates a transfer to a local SBus address that has been mapped to the VMEbus, the SBus physical address bits SBA <24:27> and the seven SBus slave select lines are used to index one 16 Mbyte segment specifier entry in the VME MMU map table.

If the segment specifier entry is valid (bit 9 of the specifier is '0'), bits <7:0> (VME Base Address) of the specifier are loaded onto the KBus address lines KA 24-31. Bit 8 of the specifier indicates whether the 16 MByte segment is D16 ('0') or D32 ('1').

The VME address is derived as follows:

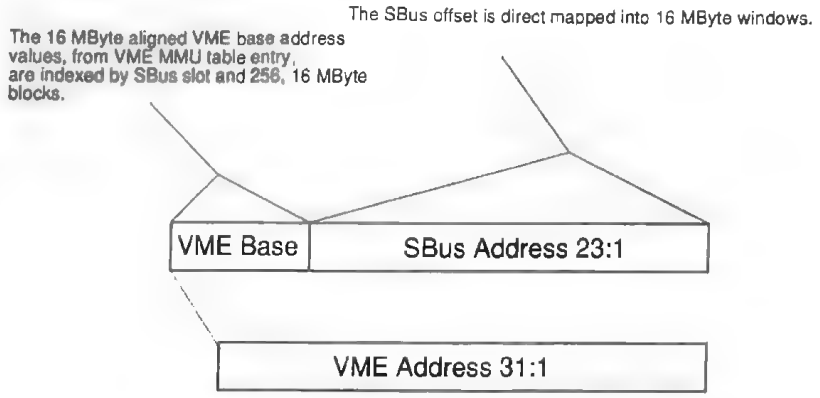


Figure 3-1. **VME Master Address Generation**

VME master accesses can be initiated by the CPU or the VSIC built-in DMA engine.

The VSIC offers the following features:

Table 3-1. **Addressing Mode**

Addressing Mode	Source Access
A64, A32, A24	DMA
A32, A24, A16	CPU

Table 3-2. **Data Transfer Mode**

Data transfer mode	Source Access
D64 (MBLT), D32 (BLT), D16(BLT)	DMA
D32, D16, D08 (EO)	CPU

The following table gives the various combinations available:

Table 3-3. **CPU Source, Data/Address Transfer Combinations**

CPU Source Transfer
A32/A24:D32
A32/A24/A16:D16
A32/A24/A16:D08 (EO)

Table 3-4. **DMA Source, Data/Address Transfer Combinations**

DMA Source Transfer
A64/A32:D64
A32: D64

Table 3-4. **DMA Source, Data/Address Transfer Combinations**

DMA Source Transfer
A32/A24:D32 (MBLT)
A32/A24:D16 (BLT)
A32/A24:D16

Master Block transfers are not supported by the CPU (programmed I/O). Only DMA-controlled transfers can make efficient use of VME block transfer mode.

All CPU accesses to VME are run in supervisor data mode as indicated in the following table.

Table 3-5. **Master Block Transfers**

Transfer Type	Address Modifier
A16	2D
A24	3D
A32	0D

DMA-sourced VME transfers may be run either in supervisor or non-privileged data mode.

Table 3-6. **DMA Source VME Transfers**

Transfer Type	Address Modifier
A24	3D or 39
A24 BLT	3F or 3B
A32	0D or 09
A32 BLT	0F or 0B
A32 MBLT	0C or 08
A64 MBLT	00

3.2.4 Addressability

The SCV64 divides the VME address space into thirty-two (32) pages of 128 MBytes each, numbered 0 to 31.

- Pages 0 to 30 (0x0000.0000 to 0x7FF.FFFF) are defined as A32:D32 by default. This means that any SBus address mapped to those pages will automatically be A32:D32.
- In page 31, VMEbus addresses will default as follows:
  - 0xF800.0000 to 0xF8FF.FFFF will default to A24:D16
  - VMEbus addresses 0xF900.0000 to 0xF9FF.FFFF will default to A23:D32
  - 0xFA00.0000 to 0xFFFE.FFFF will default to A32:D32
  - 0xFFFF.0000 to 0xFFFF.FFFF will default to A16:A16.

The SCV64 may be programmed to place the A24 region in the lower 32 MBytes of page 0. This will make page 31 A32, except for the top 64 KBytes, which is always A16:D16.

All A64 accesses must be made using the DMA controller. Also, all DMA addressing modes are independent of the USP-1 address map / VME VMMU. All DMA address transfers must be longword aligned for Block Memory Transfers (A32/A24) and double-longword aligned for Multiplexed Block Transfers (A64/A32/A24). No support for Read-Modify-Write (RMW) transfer or unaligned transfer (UAT)is provided.

3.2.5 VME MMU Address Translation

Seven 256 MByte slots in the UPA address space are dedicated to the SBus. Each slot is furthermore sub-divided into sixteen, 16 MByte segments. Any 16 MByte segment not allocated to an SBus device may be mapped to the VMEbus. The UPA base address of each 16 MByte segment is shown in section 2.7.1, Table 2-15, VME MMU Address Translation.

Mapping of SBus access cycles to the VMEbus is accomplished by the VME MMU, a memory management unit maintained as a table of mapping specifiers, one for each 16 MByte segment. The VME MMU Address Translation table starts at 0x1FF.FE00.0000

For each 16 MByte segment, the VME MMU table contains a 32-bit specifier entry that describes the segments mapping properties.

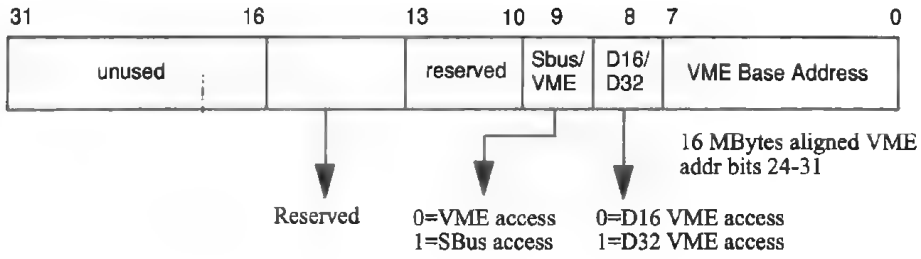


Figure 3-2. **VME MMU Map Specified Entry**

Each specifier consists of:

- Target designation: Indicates SBus, board, or VME resources (unless selection is prohibited by hardware).
- Data width: Indicates D32 or D16 mode on VMEbus accesses.
- VME base address: The 16 MByte aligned base address generated on the VMEbus (for VME accesses only). This field is used by the SCV64 to propagate A16, A24, A32 access to the VME. If the value is not F8, F9, or FF, an A32 transfer will be initiated. If the value is F8, an A24:D16 transfer will be initiated. If the value is F9, an A24:D32 transfer will be initiated. If the value is FF and SBus address bits SBA27-20 are set to 1, the SCV64 will initiate an A16:D16.

Table 3-7. **VME MMU 16MB Segment Assignments**

Segment	SBus #0 (Expansion)[a]	SBus #1 (Expansion)[b]	SBus #2 (VMEBus)	SBus #3 (MACIO 2)	SBus #4[c] (APC)	SBus #5[a] (MACIO 1)	SBus #6[a] (SLAVIO)
0	VMEbus or SBus	VMEbus or SBus	VMEbus	VMEbus	VMEbus	VMEbus	SLAVIO
1	VMEbus or SBus	VMEbus or SBus	VMEbus	VMEbus	VMEbus	VMEbus	SLAVIO

Table 3-7. **VME MMU 16MB Segment Assignments**

Segment	SBus #0 (Expansion)[a]	SBus #1 (Expansion)[b]	SBus #2 (VMEBus)	SBus #3 (MACIO 2)	SBus #4[c] (APC)	SBus #5[c] (MACIO 1)	SBus #6[c] (SLAVIO)
2	VMEbus or SBus	VMEbus or SBus	VMEbus	VMEbus	VMEbus	VMEbus	VMEbus
3	VMEbus or SBus	VMEbus or SBus	VMEbus	VMEbus	VMEbus	VMEbus	VMEbus
4	VMEbus or SBus	VMEbus or SBus	VMEbus	VMEbus	VMEbus	VMEbus	VMEbus
5	VMEbus or SBus	VMEbus or SBus	VMEbus	VMEbus	VMEbus	VMEbus	VMEbus
6	VMEbus or SBus	VMEbus or SBus	VMEbus	VMEbus	VMEbus	VMEbus	VMEbus
7	VMEbus or SBus	VMEbus or SBus	VMEbus	VMEbus	VMEbus	VMEbus	VMEbus
8	VMEbus or SBus	VMEbus or SBus	VMEbus	VMEbus	VMEbus	MACIO	VMEbus
9	VMEbus or SBus	VMEbus or SBus	VMEbus	MACIO	VMEbus	MACIO	VMEbus
A	VMEbus or SBus	VMEbus or SBus	VMEbus	VMEbus	APC	VMEbus	VMEbus
B	VMEbus or SBus	VMEbus or SBus	VMEbus	VMEbus	VMEbus	VMEbus	VMEbus
C	VMEbus or SBus	VMEbus or SBus	VMEbus	VMEbus	APC	VMEbus	VSIC
D	VMEbus or SBus	VMEbus or SBus	VMEbus	VMEbus	VMEbus	VMEbus	SCV64
E	VMEbus or SBus	VMEbus or SBus	VMEbus	VMEbus	APC	VMEbus	VMEMMU
F	VMEbus or SBus	VMEbus or SBus	VMEbus	VMEbus	VMEbus	VMEbus	IACK

a. After a system reset, SBus Slot #0 defaults to SBus.

b. After a system reset, SBus Slot #1 default to SBus.

c. Slot #4 corresponds to SBus Reference Platform slot #13. Slot #5 corresponds to SBus Reference Platform slot #14. Slot #6 corresponds to SBus Reference Platform slot #15.

3.2.6 VMEbus Slave Interface

The SCV64 offers the following features in support of slave accesses:

- VME slave access protection
- Programmable VME slave access window size
- Addressing mode A64, A32, and A24 (no A16)
- Support of Block Transfers (BLT)

- A Location monitor with a 32-deep message FIFO for inter-process communication.

The VME slave interface implemented on the USP-1 supports A64, A32 and A24 slave accesses. Block mode transfer capability is also supported. When transferring data using BLT mode on the VMEbus, the start address of the whole block of transferred data must be aligned on a 8 long word boundary, and the block size must be a multiple of 8 long words.

VMEbus slave accesses generate local SBus DVMA accesses. The slave base addresses and sizes of both the A32 and A24 address spaces are set through the VMEBAR register of the SCV64 VMEbus controller.

The slave addressing window is limited by hardware to either 8 Mbyte, 32 Mbyte, 64 Mbyte or 128 Mbyte selectable through setting jumpers JP1001 and JP1002 located on the VSIC interface.

Slave address programming information are summarized in the SCV64 User Manual. A sophisticated protection scheme could be implemented using the SCV64's APBR register. It is not supported by Solaris or VxWorks.

Every slave VME access is first validated by the SVC64. It will ignore any access outside the programmed slave VME access windows; therefore the access may time-out on the VMEbus. The SCV64 will return a bus error without initiating a local transaction if the access is protected by the APBR register or if its location monitor FIFO is full. In all other cases, SCV64 will propagate the VME slave access to the SBus through the KSIC interface and become a SBus master running a DVMA access (Direct Virtual Memory Access).

The virtual SBus address is assembled from the low order bits of the VMEbus address and the local slave window base address. The local slave base address is dependent on the configuration of the JP1001 and JP1002 jumpers. The resulting virtual SBus address is finally translated into a physical address by the IOMMU.

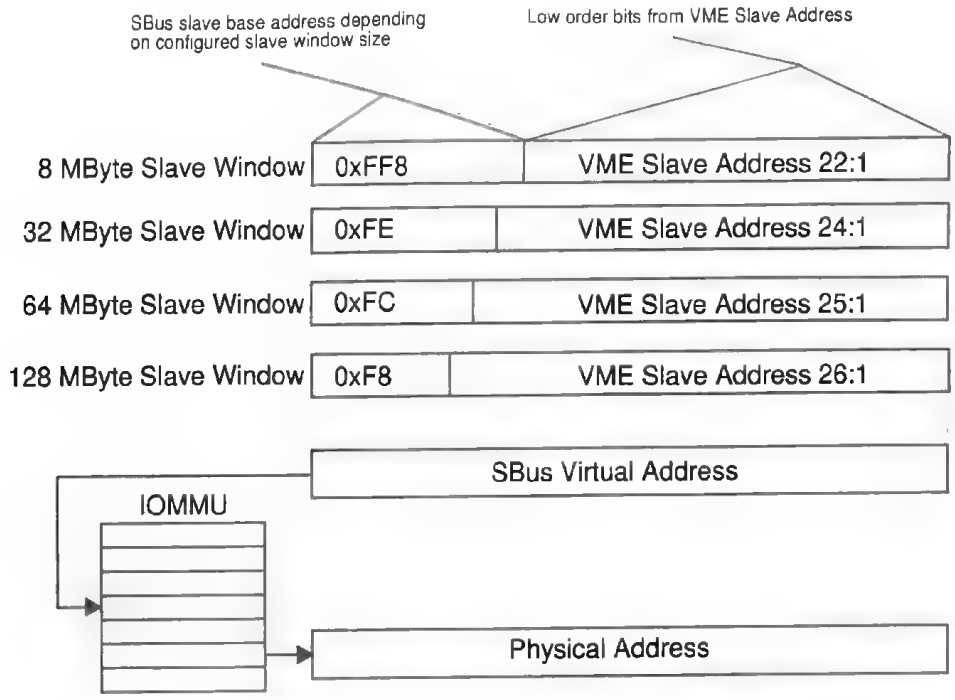


Figure 3-3. **VME Slave Address to Sbus Mapping Example**

The mechanism to convert a slave VME address into a SBus virtual address relies on the setting of the jumpers JP1001 and JP1002 on the VSIC.

Table 3-8. **Slave Window Size Configuration**

JP1001	JP1002	Slave Window Size	SBus Base Address
OFF	OFF	8 MByte (Default)	0xFF8
ON	OFF	32 Mbyte	0xFE
OFF	ON	64 Mbyte	0xFC
ON	ON	128 Mbyte	0xF8

Under Solaris 2.x, a 8 MByte slave window size is supported. The window is divided into A24 and A32 slave access:

Table 3-9. Solaris 2.x Slave Accesses

Transfer Type	Slave Window Address Range
A24	FF80 0000 to FF8F FFFF
A32	FF80 0000 to FFFF FFFF

The IOMMU translation table must be configured for the virtual DVMA address generated by a VME slave access. If there is no corresponding entry in the IOMMU, the VME slave access transfer will cause generation of a bus error by the SBus controller. Slave Read - Modify - Write (RMW) transfers are not supported.

Use of the OBP variables `vme32-slave-base`, and `vme24-slave-base` allows transparent setting of the correct VMEBAR values of the SCV64.

Caution — Any attempt to access to its own slave VME access window results in a bus error on the SBus.

3.2.7 VMEbus / SCV64 Interrupts

The VMEbus interface on the USP-1 supports all interrupt levels implemented by the VMEbus. VMEbus interrupts are converted to their corresponding SBus interrupt levels as described below. The status of each SCV64 generated interrupt is shown in the corresponding interrupt status register. After detecting a VMEbus interrupt, it is typically acknowledged by accessing the interrupt acknowledge register.

The SCV64 VMEbus controller can accept interrupt requests from three groups of interrupt sources:

- Level 7 interrupt (Abort Switch, ACFAIL, BIMODE, SYSFAIL)
- General local interrupts (LIRQ0 - LIRQ5) (only LIRQ 2, 3, 4, 5 available)
- VMEbus interrupts (VMEIRQ1 - VMEIRQ7).

SCV64 interrupts are mapped to their corresponding SBus interrupt levels as follows:

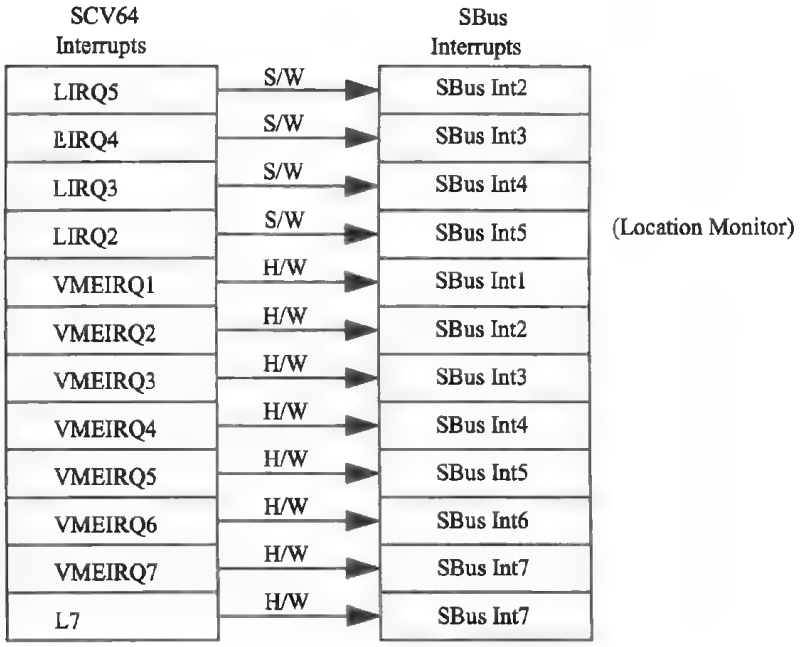


Figure 3-4. **SCV64 to SBus Interrupt Levels**

The general local interrupts are allocated as:

- LIRQ5 is connected to the Watchdog output.
- LIRQ4 is connected to the tick timer output.
- LIRQ3 is connected to the VMEINT output. This output is active whenever a VMEbus related event (i.e. DMA finished, bus error, etc.) has occurred.
- LIRQ2/KIACK is configured as KIACK. The LIRQ2 interrupt is not available externally. It is associated internally to the mailbox interrupt.
- LIRQ1 and LIRQ0 are not used and pulled to V_{cc}.

3.2.7.1 VME Interrupt Status

A 32-bit register is provided to read the status of the VMEbus interrupt request lines IRQ1 through IRQ7. The VME Interrupt Status register is located at 0x1FF FFFF 0400 and defined as follows:

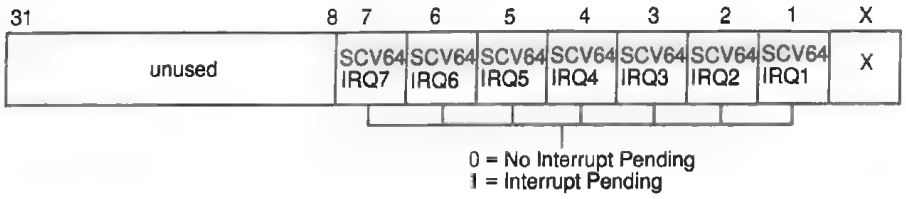


Figure 3-5. **VME Interrupt Status Register**

3.2.7.2 VMEbus Interrupt Acknowledgment

The USP-1 implements a total of 14 interrupt acknowledge registers. Seven of these registers are dedicated to local interrupt sources (i.e., DMA, location monitor, etc.). The other seven registers are assigned to VME interrupt acknowledgment.

The base address of the VME interrupt acknowledge registers is 0x1FF FF00 0000. VMEbus interrupts are generated by the SCV64 and acknowledge by the CPU reading a 32-bit (resp. 8-bit) value from the location computed by:

– register address = 0x1FF FF00 0000 (resp. 0x1FF FF00 0003) + 4 x n

where, 'n' is the level of the interrupt.

Note — If an attempt is made to acknowledge an interrupt that is not pending, a bus error is returned.

The base address of the local interrupt acknowledge registers is 0x1FF FF00 0400.

Local interrupts are acknowledged by first clearing the source of the local interrupt in the SCV64 followed by reading the corresponding interrupt acknowledge register. The location of the interrupt acknowledge register is computed by:

– register address = 0x1FF FF00 0400 + 4 x n (32-bit access) or

– register address = 0x1FF FF00 0403 + 4 x n (8-bit access)

where, 'n' is the level of the interrupt.

3.2.8 SCV64 Inter-Processor Communication / Location Monitor

A location monitor resides at the top longword (32 bits) of each A24 and A32 slave access window and is accessible from both the VMEbus and local bus. The monitor location is implemented by a Location Monitor FIFO which consists of 31 entries of 32 bits and allow the exchange of 16-bit or 32-bit messages between processes. A write access to the location monitor enters the 32-bit value into the Location Monitor FIFO. If a 16-bit word transfer was used to write the location monitor, the upper 16-bit word is set to all one. A read access to the location monitor address will result in a bus error.

While the Location Monitor FIFO contains entries, the SCV64 asserts the LMHD bit in the DCSR register and generates an interrupt to the local processor by asserting LMINT. LMINT is only negated at the end of a LMFIFO read when the FIFO is empty. LMINT may generate an SBus interrupt level 5. A write attempt will time out if the FIFO is full and no entry becomes available before the time-out counter expires.

Messages are retrieved by reading from the LMFIFO register. Attempts to read from an empty FIFO will result in undefined data being returned

Caution — If the LMFIFO is read while it is empty and a message arrives from the VMEbus, then unstable data may be put on the local bus. Do not read the LMFIFO unless the LMHD bit is set and LMINT is asserted.

The local CPU may write to its own location monitor through its slave address window on the VMEbus. The SCV64 address decoder responds by redirecting the message to the LMFIFO.

3.3 Overview of VME Interface Under Solaris 2.x

Great effort has been expended to achieve complete compatibility between Sun VME implementations and the device driver interface implemented for the USP-1 board under Solaris.

This section is primarily concerned with the implementation and programming of USP-1 specific features that are not common to the UltraSPARC-I family of workstations (SPARCstation 5). It assumes that the reader is familiar with the Sun4u architecture and concepts of the VMEbus and the device drivers.

The VMEbus interface of the USP-1 is transparently implemented under Solaris 2.x. The software interface is fully compatible with generic Sun4u VME architecture systems. Any VME device driver that conforms to the appropriate Solaris Device Driver Interface will run unmodified on Themis platforms.

Themis Computer has developed a number of device drivers to provide the necessary software interface needed by system programmers. The drivers can be classified into three categories:

- VMEbus Nexus drivers
- Utility drivers
- Sample drivers.

The drivers are described in the following subsections. Themis also provides online manual pages for all the drivers included in the software interface. These manual pages include more detailed information on the drivers.

3.3.1 VMEbus Nexus Driver

The VME driver forms the core of the software interface provided by Themis Computer. VME is a bus nexus driver that encapsulates all the architectural features of supporting the Themis USP-1 board on Solaris systems. This driver is configured to be permanently loaded as the bus nexus driver for the VMEbus.

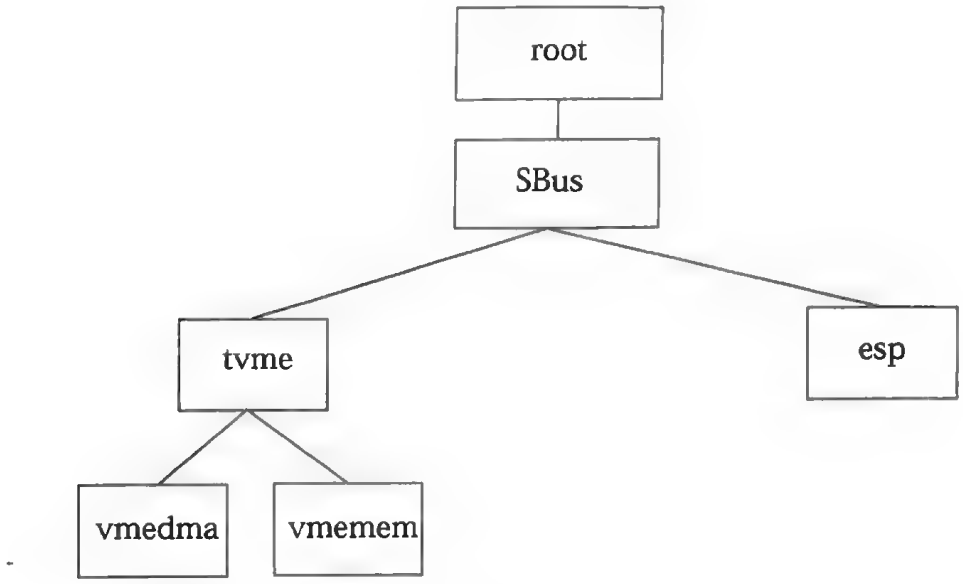


Figure 3-6. **Nexus Driver Directory Structure**

Any driver that specifies its class or parent as vme will be attached to the vme driver. The vme driver provides support for all VMEbus operations, including VME interrupt processing, Direct Virtual Memory Access (DVMA) on the VMEbus, mapping of registers on the VMEbus to kernel space, mapping of VMEbus memory to user address space through mmap(), etc.

The VME driver recognizes the standard configuration file for VME device drivers. (see the vme(4) man page). The configuration of any VME device driver specifies the VMEbus interrupt level and the interrupt vector. The mapping of the VMEbus interrupt level to the SPARC interrupt priority level is as follows

Table 3-10. **VMEbus Interrupt Mapping**

System	Interrupt Level							
VMEbus		1	2	3	4	5	6	7
SPARC ip	1	2	3	5	7	9	11	13

3.3.2 Utility Drivers

These drivers are a standard part of the software interface provided by Themis Computer. These drivers provide a useful interface to system programmers who wish to utilize the different features of the VMEbus architecture.

The VME device driver provides the standard /dev devices for accessing the VMEbus from user processes:

Table 3-11. **VME Device Drivers**

Device File	Address Size	Data Transfer Size
/ dev / vme32d32	32	32
/ dev / vme32d16	32	16

Table 3-11. **VME Device Drivers**

Device File	Address Size	Data Transfer Size
/ dev / vme24d32	24	32
/ dev / vme24d16	24	16
/ dev / vme16d32	16	32
/dev / vme16d16	16	16

`read()`, `write()` and `mmap()` calls are supported and no special `ioctl()` calls are required to configure the interface.

The SCV driver provides access to features of the SCV64 chip, including the Direct Memory Access engine. The SCV driver is configured as a child of the VME (7) driver, and will function only on Themis platforms that feature the Tundra SCV64 VMEbus Controller. The driver supports 16-byte aligned DMA transfers between the main memory and the VMEbus. The user can specify the direction of the transfer and the VME address space that is involved in the transfer.

3.3.3 Sample Drivers

The software interface provided for the USP-1 platform fully supports standard VMEbus device drivers written for Solaris environments. It is the intention of Themis Computer to fully support users who wish to write their own VME device drivers that would function on USP-1 platforms. To aid these users and to illustrate the specific features of the VMEbus architecture, Themis provides a number of sample device drivers. Themis provides the complete source code and the necessary configuration files for these drivers.

- **vmeintr:** is a sample driver provided by Themis Computer to illustrate the vectored interrupt mechanism of the VMEbus. The vmeintr driver relies on the driver configuration files to specify the interrupts it should handle. Each instance of the driver registers the interrupts with the system. Through the ioctls implemented by the driver, the user can generate interrupts and send them to the appropriate driver instances. The interrupt generator and the interrupt receiver need to run on different computers.
- **vmedvma**: is a sample driver provided by Themis Computer to illustrate the use of Direct Virtual Memory Access within a device driver. The user can ask the driver to allocate a DMA region on the VMEbus. The driver allocates all the necessary resources to support a DVMA transfer to this region. The driver also implements mechanisms by which the user program can write to or read from the DVMA region.

3.4 Installing the USP-1 Software

The software interface for USP-1 is distributed as a software package for Solaris 2.x systems. The software package is named THEMISvme and can be installed and removed like other standard Solaris software packages, by using the `pkgadd` and `pkgrm` commands.

3.4.1 Installing THEMISvme

The THEMISvme package is distributed on standard media and can be installed directly from the media. To install the package, place the installation media in the appropriate slot and execute this command:

```
# pkgadd -d <media name>
```

where <media name> is the name of the media on your system.

The pkgadd command will copy the contents of the package to the appropriate directories and perform the necessary installation. After the command completes, the system has to be rebooted for the new software to be operational.

The user may choose to install these drivers by executing a script provided for that purpose. The sample drivers are not required for the normal operation of the USP-1 system.

3.4.2 Detailed Installation

During the course of installation, pkgadd will prompt the user for input on optional configuration. The first prompt will request the user to choose the packages to be installed:

usp1_vme# pkgadd -d THEMISvme.file

```
The following packages are available:

  1   THEMISvme      Themis VME Drivers for SPARC USP-1 (Solaris 2.5)

                       (sparc) USP-1 Release 1.0

Select package(s) you wish to process (or 'all' to process all packages).
(default: all) [?,??,q]:


Processing package instance <THEMISvme> from </export/drivers/vme/
THEMISvme.fil

>

Themis VME Drivers for SPARC USP-1 (Solaris 2.5)

(sparc) USP-1 Release 1.0

Themis Computer


You are about to install the vme nexus driver for Solaris 2.5
```

```
to use the VMEbus interface on the Themis USP-1.

The package replaces the functionality of the vmemem and
sbusmem drivers from the original Sun distribution.
However current versions of the files will be retained.
Please ensure that the sbusmem and vmemem drivers are not currently used.

After the installation, the system has to be rebooted to complete
the installation. If the system cannot be rebooted at this time,
you may abort the installation of the package now.
Continue installation? [n] [y,n,?,q] y

Where should the driver objects be installed
[/platform/sun4u/kernel/drv] [?,q]
Using </> as the package base directory.
## Processing package information.
## Processing system information.
   4 package pathnames are already properly installed.
## Verifying disk space requirements.
## Checking for conflicts with packages already installed.
## Checking for setuid/setgid programs.

This package contains scripts which will be executed with super-user
permission during the process of installing this package.
Do you want to continue with the installation of <THEMISvme> [y,n,?] y
```

```
Installing Themis VME Drivers for SPARC USP-1 (Solaris 2.5) as
<THEMISvme>

## Executing preinstall script.

## Installing part 1 of 1.

[ verifying class <none> ]

[ verifying class <devlink> ]

/platform/sun4u/kernel/drv/sbvmemem

/platform/sun4u/kernel/drv/sbvmemem.conf

/platform/sun4u/kernel/drv/themvme

[ verifying class <drv> ]

## Executing postinstall script.

Reboot client to install driver.

Note: major number maximum based on server, not client

Reboot client to install driver.

Note: major number maximum based on server, not client


It will be necessary to do a reconfiguration reboot after driver

installation. As soon as possible, execute 'reboot -- -r' to reboot.

The new vme drivers will not be functional till a reboot is done.


Installation of <THEMISvme> was successful.

usp1_vme#
```

3.4.3 Removing THEMISvme

If you want to remove the software interface of USP-1 for any reason, you can execute the pkgrm command to remove the THEMISvme package:

```
# pkgrm THEMISvme
```

pkgrm will remove all the files contained in the package. The original contents of VME drivers before the package was installed will be restored.

3.4.4 Installing THEMISvme On Diskless or Dataless Clients

Currently, most of the software interface contained in THEMISvme can be installed on diskless and dataless client systems. On these clients, the VMEbus drivers will be copied and installed correctly.

3.5 Writing Programs for the VMEbus

The software interface of the USP-1 is transparently implemented under Solaris 2.x. The software interface is fully compatible with generic Sun4u VME architecture systems. Any VME device driver that confirms to the appropriate Solaris Device Driver Interface will run unmodified on Themis platforms.

This section is intended for system programmers who are not familiar with the Solaris architecture and the concepts of the VMEbus. The guide discusses the concepts of Solaris drivers and VMEbus devices as they pertain to the Themis USP-1 product. The section is intended as a general introduction of the basic concepts of using the USP-1 product under Solaris.

3.5.1 Solaris 2.x Device Hierarchy

Solaris 2.x systems support architecture independence of devices by using a layered approach. Each node in the tree structure has a specific device function. Standard devices are associated with leaf nodes. The drivers for these devices are called leaf drivers. The intermediate nodes in the tree are associated with buses, like SBus, VMEbus etc. These nodes are called bus nexus nodes and the drivers associated with them are called bus nexus drivers. The bus nexus driver encapsulates all the architectural dependencies of a particular bus. The leaf driver only needs to know the kind of bus it is connected to.

The device tree for USP-1 systems is shown below.

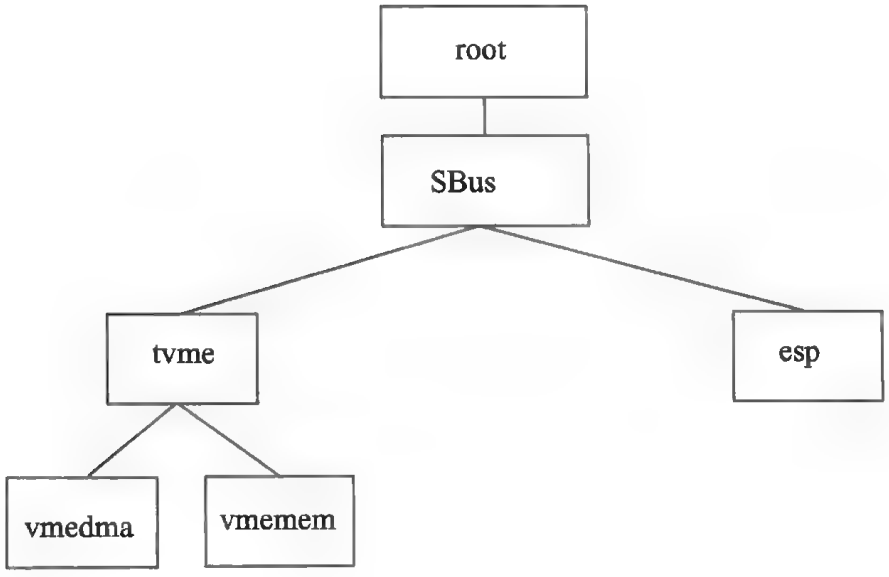


Figure 3-7. **USP-1 Device Tree**

In this diagram, root, tvme, and SBus are bus nexus drivers. The tvme driver encapsulates all the details of the implementation of the VMEbus interface. The vmemem driver, which is a leaf driver, only needs to know that it is connected to the VMEbus.

Solaris, like other UNIX systems, represents devices as special files in the file systems. These files are advertised by the device driver and are maintained by the drvconfig(1M) program. Usually the special files are created under the /devices directory and represent the hardware layout of a particular machine. sysdef(1) and prtconf(1) can be used to view the internal device tree. Symbolic links are created from the /dev directory to the special files in the /devices directory. User programs normally use the files from the /dev directory.

VMEbus devices are not self-identifying i.e. they do not provide information themselves to the system. Drivers for VMEbus devices need to have a probe() entry so that the kernel can determine if the device is really present. Additional information about the device must be provided in a hardware configuration file. See driver.conf(4) and vme(4) for more information.

VMEbus interrupts are vectored. When a device interrupts, it provides the priority as well as an interrupt vector. The kernel uses the information to uniquely identify the interrupt service routine to be called. The hardware configuration file must specify each priority-vector pair that the device may issue.

3.5.2 Configuring the Software Interface of USP-1

The USP-1 board can be accessed in a slave mode from another board on the VMEbus chassis. This type of access requires specifying the slave base address and the size of the slave access region of the USP-1 board Distinct values can be specified for 24 bit addresses and 32 bit addresses. On a VMEbus chassis with multiple boards, each processor board is required to have a non-overlapping region for slave mode access. The OBP variables vme32-slave-base and vme24-slave-base can be used to set the slave base addresses of a USP-1 board.

3.5.3 Accessing the VMEbus from OBP

This section briefly explains some ways of to access the VMEbus from OBP. For a description of OBP commands themselves, please refer to the document named OpenBoot Command Reference and to Chapter 11, "Open Boot PROM Extension Commands."

3.5.3.1 Accessing VMEbus memory

The OBP does not initialize the VME MMU. To access the VME memory using the USP1 OBP, it is necessary to first boot Solaris. Solaris will initialize the VME MMU and allow access to VME memory. After the VME MMU has been initialized the VME memory may be accessed from OBP. The following example provides sample device driver information for Solaris 2.x and demonstrates an access to the VME Memory from OBP. In Solaris 2.x:

In sampledrv.conf:

```
name="sampledrv"  class="vme"  reg=0x4d,  0x72000000,  0x100000  interrupts=2,
0x82;
```

In sampledrv.c:

```
caddr_t reg;
```

```
...

ddi_map_regs(devi, 0, &reg, 0, 0x400)

[returns reg-0x50508000 which is the vaddre of teh mapped area.]
```

Now, exit Solaris and enter into OBP.

```
ok 50508000 map?

VA:50508000

G:0 W:0 P:1 E:1 CV:0 CP:0 L:0 Soft1:c PA[40:13]:ff97800 PA:1ff2f000000

Diag:0 Soft2:100 IE:0 NFO:0 Size:0 V:1

PA:1ff2f000000
```

Then, access the VME memory using the following command:

```
1ff2f000000 2f 100 memmap 100 dump
```

	\/	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f	v123456789abcdef
FFFA6000	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	.....
FFFA6010	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	.....
FFFA6020	44	44	44	44	44	44	44	44	44	44	44	44	44	44	44	44	DDDDDDDDDDDDDDDD
FFFA6030	44	44	44	44	44	44	44	44	44	44	44	44	44	44	44	44	DDDDDDDDDDDDDDDD
FFFA6040	44	44	44	44	44	44	44	44	44	44	44	44	44	44	44	44	DDDDDDDDDDDDDDDD

3.5.4 Accessing the VMEbus from Solaris

This section describes the many features of the software interface provided by Themis Computer. The features provide the system programmer with powerful ways of accessing the VMEbus from the Solaris Operating System environment. Themis Computer has developed many sample programs that illustrate the ways of using the software interface provided. These programs are available with the source code and can be used to test the configuration of a VME system. They can also be used as example programs for the use of the different features provided by the software interface.

3.5.4.1 Using Read/Write

The vmemem driver from Themis Computer enables the user to access any VMEbus address from a user program. Depending on the address space used by the particular VMEbus card, different devices need to be used.

Table 3-12. VME Table

File	Address Size	Data Transfer Size	Physical Address
/dev/vme16d16	16 bits	16 bits	0x0-0xFFFF
/dev/vme24d16	24 bits	16 bits	0x0-0xFFFFFF

Table 3-12. VME Table

File	Address Size	Data Transfer Size	Physical Address
/dev/vme32d16	32 bits	16 bits	0x0-0xFFFFFFFF
/dev/vme16d32	16 bits	32 bits	0x0-0xFFFF
/dev/vme24d32	24 bits	32 bits	0x0-0xFFFFFF
/dev/vme32d32	32 bits	32 bits	0x0-0xFFFFFFFF

As an example, to access a VME card that supports 24-bit addresses and 32-bit data transfers, the user needs to use the file /dev/vme24d32.

Using /dev/vmeXXdYY files is very similar to using normal files; the user program opens the file, uses the memory address as an offset to lseek to and does a read or write operation at the offset. The 'offset' must fall within the physical address range of the file that was opened; also there should be a VMEbus device that would respond to the address.

To access 16 bytes at the VMEbus address 0xa0000, the following code snippet can be used:

```
..
int fd;
int vme_base;
char buffer[16];
if ( ( fd = open( "/dev/vme24d32", O_RDWR)) < 0)
{
perror("In opening file /dev/vme24d32");
return(-1);
}

vme_base = 0xa0000;
if ( lseek ( fd, vme_base, SEEK_SET) < 0)
{
perror("In lseeking to 0xa0000");
close(fd);
return(-1);
}
```

```
if ( read ( fd, buffer, sizeof(buffer) ) != sizeof(buffer) )
{
perror("In reading 16 bytes");
close(fd);
return(-1);
}
.
.
```

In the above example, if there is no device on the VMEbus at address 0xa0000, the VMEbus operation will time-out and the program would receive a SIGBUS signal.

The vmeXXdYY files can also be used to quickly check the proper installation of a VME board. e.g. After installing a VME memory board at address 0x72000000, the user can use od(1) to look at the memory:

```
> od -x /dev/vme32d32 +0x72000000
```

If the od command displays '0x72000000' and nothing else, it indicates that there is no VME device at the address 0x72000000.

3.5.4.2 Using mmap

A faster way to access VMEbus space is to use the mmap(2) system call. mmap() establishes a mapping between the user process's address space and a memory object represented by an open file. The /dev/vmeXXdYY files can be used for mmap() as well. e.g. To access a region of length 1000 bytes in the VMEbus space, starting from location 0x500, the following code segment can be used:

```
.
int vme;
caddr_t pageptr;
.

if ((vme = open("/dev/vme32d32",O_RDWR)) < 0) {
        perror("open error on VME file");
        return;
```

```
}

if ((pageptr = mmap((caddr_t)0L, roundup( 1000, PAGESIZE),
PROT_READ|PROT_WRITE, MAP_SHARED, vme,
(off_t)(0x500 & PAGEMASK ))) == (caddr_t)-1) {
        perror("mmap error");
        return;
}
.
```

Note that the 'off' parameter to mmap() is constrained to be aligned at a page boundary. The 'len' parameter does not have a size or alignment constraint. The success of mmap() does not imply that the specified range of VMEbus is valid and accessible. If the region specified by [off, off+len] is not a valid VMEbus address region, access to the region will result in a SIGBUS signal. It is often convenient to catch this signal and take some action. The following code snippet sets up a catching mechanism for SIGBUS and SIGSEGV:

```
.
void busError(int sig_num )
{
printf("\nBus Error: received signal :%d\n", sig_num);
return(0);
}


.
.
main()
.
.
signal(SIGBUS, busError);
signal(SIGSEGV, busError);
.
```

3.5.4.3 Slave Mode Access to Another VME Board

Memory on the USP-1 board can be accessed from another board on the VMEbus chassis. This type of access is called slave mode access. Slave mode access under Solaris is tied to the concept of Direct Virtual Memory Access (DVMA). DVMA is a facility present on SPARC hardware that allows a device driver to specify virtual addresses rather than physical addresses for a DMA operation. The kernel maintains a map that specifies the correspondence between the DVMA address and the physical memory.

In a typical operation, the master issues an address on the VMEbus; this address falls within the slave address range of another board on the chassis. The VME controller on the slave board then converts the VME address into a SBus virtual address. The VME controller then performs a DVMA operation on the SBus to access the physical memory location. The SBus virtual address generated by the VME slave access must be configured in the IOMMU translation table. Themis Computer provides a driver that performs this configuration; the driver allocates a DVMA region and sets it up for VMEbus slave access at a particular VMEbus address. The master board simply uses this VMEbus address and generates a slave access operation on the VMEbus. The system that allocated the DVMA memory can also access the region by using special ioctl calls provided by the driver. Please refer to vmedvma(7) for more information on the driver.

The following code segment sets up a memory region of size 8192 bytes for DVMA access:

```
.
#include <themis/vmedvma.h>
.
.
.
int fd;
struct vme_dvmacopy dma_req;
  /* open the device */
if ((fd = open("/dev/vmedvma", O_RDWR))<0)
{
     perror("can't open /dev/vmedvma");
     exit(1);
}
/* allocate a DVMA region */
dma_req.size = 8192;
if (ioctl(fd, VDMA_ALLOCATE, &dma_req) != 0)
{
```

```
        perror("In allocating DVMA");

        close(fd);

        return(-1);

    }

    printf("Allocated a DVMA region id : %d, at virtual address : %x\n",
    dma_req.id, dma_req.addr);

    .

    .

    .
```

The 'virtual address' output by this program can be used by other boards on the VMEbus chassis to gain access to this DVMA region. e.g. If a second board on the same chassis needs to access this region, one can execute this command:

```
> od -x /dev/vme24d32 {slave_address}
```

where {slave_address} is the virtual address output by the program.

The DVMA region can be accessed from the same system by invoking the ioctl() calls implemented by the vmedvma driver. The following code segment fills a DVMA region with zeros, starting from offset 0x500.

```
    .

    .

    .

    struct vme_dvmacopy dma_write;

    char * buffer;

    .

    .

    buffer = malloc( 4096);


    dma_write.id = 1;

    dma_write.addr= (caddr_t) buffer;

    dma_write.size= 4096;

    dma_write.offset= 2048;
```

```
if (ioctl(fd, VDMA_WRITE, &dma_write) != 0)
{
perror("In initializing DVMA region");
close(fd);
return(-1);
}
.
.
.
```

3.6 Writing VME Device Drivers

A device driver is a kernel module responsible for managing low-level I/O operations for a particular hardware device. Some device drivers manage 'fake' devices that exist only in software. A device driver contains all the device-specific code to communicate with a device. Like the system call interface protects application programs from the specifics of the platform, the device drivers protect the kernel from the specifics of the devices. A device driver also provides a standard I/O interface to the rest of the system; i.e., a user program should be able to open a 'device file' and issue (in most cases a subset of) standard system calls to the file.

The VMEbus interface of Themis Computer's USP-1 is transparently implemented under Solaris 1.x (SunOS 4.1.3) and Solaris 2.x. The software interface is fully compatible with generic Sun4m VME architecture systems. Any VME device driver that confirms to the appropriate Solaris Device Driver Interface will run unmodified on Themis platforms.

This guide is intended to highlight the VMEbus specific features that influence the design of drivers for VMEbus devices. It tries to provide basic information, and when needed, refers to sources for more detailed information on specific concepts. A full introduction to the principles of writing device drivers is beyond the scope of this guide. For information on writing device drivers for Solaris 1.x systems, please consult the document "Writing Device Drivers". The book "SunOS 5.3 Writing Device Drivers" has extensive information on writing device drivers for Solaris 2.x systems.

Themis Computer has developed a number of sample device drivers that illustrate the concepts covered in this chapter. Please see Chapter 8, Sample Device Drivers, for more information on the sample device drivers. These drivers, along with user level programs that interface to them, are provided in source code format.

The last section in the guide discusses the situations where developing a custom device driver may not be necessary. The section discusses mechanism by which most device operations can be performed from the user level.

In its current version, this guide focuses on the software interface provided under Solaris 2.x. However except some sections, the concepts in this guide apply to Solaris 1.1.1 systems also.

3.6.1 Configuration Files for VME Device Drivers

Driver configuration files pass information about device drivers and their configuration to the system. Since VMEbus devices are not self-identifying, drivers for these devices need to use driver configuration files to inform the system that the device hardware may be present. The configuration file also must specify the device addresses on the VMEbus and any interrupt capabilities that the device may have. Please see driver.conf(4) and vme(4) for more details on the configuration files.

The syntax of an entry in the configuration file is:

```
name="driver name" class="vme" reg="...." interrupts="...";
```

Specifying class as "vme" indicates to the kernel that the device driver is for a VMEbus device and should be attached to the bus nexus driver for VMEbus. On USP-1 systems, the bus nexus driver is named vme.

Two common properties of interest to VME device drivers are the reg and interrupts properties. The reg property is an array of 3-tuples: address space, offset and length. Each 3-tuple describes a contiguous resource (registers) that can be mapped. The value of the address space is derived from the following table:

Table 3-13. **Address Space**

Address Space	Value
A16:D16	0x2D
A24:D16	0x3D
A32:D16	0xD
A16:D32	0x6D
A24:D32	0x7D
A32:D32	0x4D

As an example, a device that supports A24D32 and has a set of registers that are 32-byte long and start at address 0xee00, the following 3-tuple can be used: 0x7d, 0xee00,32. Multiple register sets are separated by commas.

VMEbus supports vectored interrupts. The interrupts property in the driver configuration file specifies all the interrupts that the device may generate. The interrupts property is an array of 2-tuples: VMEbus interrupt level, VMEbus vector number. e.g. for a device that generates interrupts at VME level 3 with a vector of 0x81, this two-tuple can be used: 2, 0x81. The VMEbus interrupt level has a value between 1 and 7, and the vector a value between 0 and 255.

All VMEbus device drivers must provide reg properties. The first two integer elements of this property are used to construct the address part of the device name under /devices. Only devices that generate interrupts need to provide interrupts properties.

It is to be noted that the DDI/DKI functions are capable of handling the VMEbus specific features, provided that the hardware configuration file is correct. e.g. if the hardware configuration file correctly defines a register set, the ddi_map_regs() function can be used to map the register set into kernel address space, without any extra effort from the driver code.

3.6.2 Probing devices

Since VME devices are not self-identifying, drivers for these devices are required to have a probe entry point. Usually the probe entry point tries to map the registers and then attempt to access the hardware using any of the peek and poke routines. The probe entry point should not initialize the device in anyway; it also should not initialize any of the software state.

Mapping registers from VMEbus space does not need any special considerations. The ddi_map_regs() function call can be used to map the registers; this function will take care of the VMEbus address space that the register set is in.

The following code segment is an example of mapping a register set and 'carefully' accessing a byte in the set.

```
/*
 * xxprobe          - probe for this device
 */
static int
xxprobe(dev_info_t *dip)
{
     struct my_reg_str *regs; /* Device registers */
     caddr_t kaddr;        /* mapped register address */
     int retval;           /* return value */

     /*
      * If we're self-identifying, then we're here.
      */
     if (ddi_dev_is_sid(dip) == DDI_SUCCESS)
          return DDI_PROBE_SUCCESS;

     /*
```

```
        * Probe the device by mapping the registers, and reading the
        * command register and the parm7 register...This assures that
        * *something* is there. The attach routine will try to run
* more rigorous checks. So this simple check is all we
* really need here.
*/
    if ((retval = ddi_map_regs(dip, 0, &kaddr, 0, 0)) != DDI_SUCCESS)
{
            cmn_err(CE_WARN,"xx%d(probe): can't map my registers,
error%d\n",
            ddi_get_instance(dip), retval);
        return DDI_PROBE_FAILURE;
    }

    regs = (struct my_reg_str *)kaddr;
    retval = ddi_peek1(dip, (long *)&regs->command, (long *)0)
            == DDI_SUCCESS ? DDI_PROBE_SUCCESS : DDI_PROBE_FAILURE;
    if (retval == DDI_PROBE_SUCCESS)
        retval = ddi_peek1(dip, (long *)&regs->parm7, (long *)0)
                            == DDI_SUCCESS ? DDI_PROBE_SUCCESS :
DDI_PROBE_FAILURE;
ddi_unmap_regs(dip, 0, &kaddr, 0, 0);
return retval;
}
```

3.6.3 Registering Interrupts

VMEbus interrupts are vectored. When a device interrupts, it provides the priority as well as an interrup. vector. The kernel uses the information to uniquely identify the interrupt service routine to be called. The hardware configuration file must specify each priority-vector pair that the device may issue. Once this has

been done, the driver code can call the ddi_add_intr() function to register the interrupts. On return from this function, the fourth argument contains a pointer to useful information like the VMEbus interrupt priority and the VMEbus vector number.

The following code segment provides an example of registering an interrupt with the kernel; the code first gets the number of interrupt specification in the configuration file; then it registers an interrupt handler for each interrupt specification.

```
static int

xxattach(dev_info_t * dip, ddi_attach_cmd_t cmd)

{

struct xxstat * xsp;

int nbint,i;

.


if ( cmd != DDI_ATTACH)

return (DDI_FAILURE);

.

.

/* get the number of interrupt definitions */

if (ddi_dev_nintrs(devi, &nbint) == DDI_FAILURE)

{

cmn_err(CE_WARN,"xx: No interrupts property.");

return (DDI_FAILURE);

}


for ( i=0; i < nbint; i++)

{

/* Register first with a null handler so that the interrupt

 * routine doesn't grab the mutex
```

```
 */
if ( ddi_add_intr(dip, i, &xsp->iblock_cookie[i],NULL,
( (u_int (*) (caddr_t) nulldev, NULL) != DDI_SUCCESS)

{
cmn_err(CE_WARN,"xx: cannot register interrupt.");
return (DDI_FAILURE);
}

7* Initialize the mutex now. */
mutex_init ( &xsp->mu, "xx mutex", MUTEX_DRIVER,
(void *) xsp->iblock_cookie[i]);

/* Remove the interrupt and register again with the correct
 * interrupt handler.
 */
ddi_remove_intr ( dip, i, xsp->iblock_cookie[i]);
if ( ddi_add_intr(dip, i, &xsp->iblock_cookie[i],
&xsp->idevice_cookie[i], xxintr, NULL) != DDI_SUCCESS)

{
cmn_err(CE_WARN,"xx: cannot register interrupt.");
return (DDI_FAILURE);
}

cmn_err (CE_NOTE,"Registered interrupt %d with priority %d and vector
0x%x",
```

```
i, xsp->idevice_cookie[i].idev_priority,
xsp->idevice_cookie[i].idev_vector);
}
.
.
```

Themis Computer provides a sample device driver, vmeintr, to illustrate the use of interrupts in device drivers. This driver is provided along with the source code and a sample configuration file. The reader may wish to study the source code for this driver to gain a better understanding of handling interrupts in device drivers.

3.6.4 Allocating DVMA Space

Direct Virtual Memory Access is a facility present on SPARC hardware that allows a device to specify virtual addresses rather than physical addresses for a DMA operation.

The kernel maintains a map that specifies the correspondence between the DVMA address and the physical memory. It is the responsibility of the device driver to set up the memory region for DVMA access.

Setting up a DVMA region is a two-step process: allocating memory and allocating DMA resources for this memory region. The following code segment is an example of setting up a DVMA region of size 'len' bytes.

While allocating memory and setting it up for DVMA, the driver can describe the capabilities of the DMA engine to the kernel by using a ddi_dma_lim_t structure. On USP-1 systems, the capabilities are as follows:

```
.
static ddi_dma_lim_t limits =
{
0x0,
0xffffffff,
0xffffffff,
07f,
0x1,
0
};
.
```

```
.
caddr_t kmem_base;
int req_id, real_len, req_len;
ddi_dma_handle_t handle;
ddi_dma_cookie_t cookie;
dma_req_t *reqp;
.
.
req_len = len;
real_len = 0;

/* allocate memory */
     if (ddi_mem_alloc (xsp->dip, &limits,
                  (size_t)req_len, 0, &kmem_base,
                  (uint *) &real_len) != DDI_SUCCESS)
     {
          cmn_err (CE_NOTE, "ddi_mem_alloc failed.");
          return (ENOMEM);
     }

/* set it up for DMA access */
     if (ddi_dma_addr_setup (xsp->dip, (struct as *) NULL,
          kmem_base, real_len,
          DDI_DMA_RDWR, DDI_DMA_DONTWAIT, 0, &limits,
          &handle) != DDI_DMA_MAPPED)
     {
          ddi_mem_free (kmem_base);
```

```
            cmn_err (CE_NOTE, "ddi_dma_addr_setup failed.");
            return (EFAULT);
        }

/* extract the virtual address of the memory region.
 * this can be obtained from the DMA cookie structure.
 */

    if (ddi_dma_htoc (handle, 0, &cookie) != DDI_SUCCESS
     -  ||
        ddi_dma_sync (handle, 0, real_len, DDI_DMA_SYNC_FORDEV)
        != DDI_SUCCESS)
    {
        ddi_dma_free (handle);
        ddi_mem_free (kmem_base);
        cmn_err (CE_NOTE, " ddi_dma_sync() failed\n");
        return (EFAULT);
    }

cmn_err ( CE_NOTE,"Virtual address of memory  is 0x%x",kmem_base);
cmn_err ( CE_NOTE,"Virtual address of the DVMA region is 0x%x",
(caddr_t) cookie.dmac_address );
.
.
.
```

The address contained in the 'DMA cookie' is the 'virtual address' of the DVMA memory region and can be used to access the memory region over the VMEbus. e.g. if the 'virtual address' printed by the second cmn_err statement is 0x588, you can access this memory region from another system on the VMEbus chassis by using this command:

```
# od -x /dev/vme24d32 0x588
```

3.6.5 Mapping VMEbus Space

On some occasions, it would be convenient to map a chunk of VMEbus address space into the system's memory. From the user level, this can be done by doing a mmap() operation on the appropriate /dev/vmeXXdYY file. From the driver level, mapping VMEbus address space can be done by using the ddi_map_regs() function. As an example, to map two pages of VMEbus space starting at location 0x66600000, the following code segment can be used:

```
if (ddi_map_regs(dip, 0, &map_space, 0x66000000, (off_t) 8192) !=
DDI_SUCCESS)

{

return (EFAULT);

}
```

On return from the function, map_space contains the base address of the kernel virtual region. There are many caveats to using ddi_map_regs() to map a portion of VMEbus address space.

1. An A32:D32 device can only map a chunk from A32:D32 address space.
2. The address space indicated by [offset, offset+len] must be a valid region on the VMEbus.
3. Access to the mapped region should be localized. Subsequent access to locations that are far apart will result in many page faults, particularly on regions of larger sizes.

It is prudent to use the ddi_peek() and ddi_poke() routines to access a region mapped using ddi_map_regs().

3.6.6 Driving Devices Without Writing Device Drivers

As mentioned before, device drivers protect the rest of the kernel from the specifics of hardware devices. Most hardware devices would necessitate the development of a device driver before they can be used on a computer system. However the features of the VMEbus and the design and implementation of the Themis Computer software interface for VME alleviate the need for specialized device drivers for most simple VME devices. This provides a tremendous advantage to programmers writing applications for these simple devices.

In Solaris 2.x device drivers are dynamically loadable. This avoids the need to re-link the kernel and reboot the system whenever a driver is modified. However, a fault in the driver could still cause the kernel to panic, resulting in long downtimes for the machine. Further, an abnormal shutdown of the machine, as in the case of kernel panic, could cause the disk to be corrupted and some important files to be lost. These problems can be avoided if the program driving the device executes in user mode instead of kernel mode.

This section discusses some ways by which an user program can do device operations that are traditionally performed by a device driver. The section is intended to clue the programmer on to the possibilities of accessing devices from user level. The reader is encouraged to go through the source code of the sample drivers to gain an insight into how they can be used for user-level access.

In simplistic terms, a device can be viewed as a collection of registers that are addressable. Almost all of these registers can be either read from or written to. Some of them allow both read and write access. Access to many of the registers results in effects like resetting the value of the register, clearing the interrupt etc. Some registers require a particular sequence of accessing them. Some registers have specific data widths that can be used to access them. Though there are some devices that have some very peculiar behavior, most devices can be modeled as a collection of registers. Such devices, if they are VMEbus devices, can be programmed and used without the need for a specialized device driver.

As mentioned before, VMEbus addresses are not distinguishable from memory addresses. The vmemem drivers from Themis Computer, which is the device driver for the /dev/vmexxdyy files, enables an user program to access any address in the VMEbus space.

This feature can be used to access the registers of a device from a user program. The user program needs to open the appropriate /dev/vmeXXdYY file, lseek to the address of the device registers and do a read or write operation. The vmemem driver performs the read/write operation 'carefully' by using the ddi_peek() and ddi_poke() calls. If the user program opens the appropriate file and the read/write access meets the alignment criteria of the device, the device registers can be easily programmed from the user program.

.e.g To access a device that supports 32 address bits and has 32-bit data, with the registers starting at offset 0x7e000000, the user program needs to open /dev/vme32d32 and lseek to offset 0x7e000000. Please note that the registers of this device could be 8-bit wide, 16-bit wide or 32-bit wide. If the read/write call is issued with the appropriate length, the vmemem driver will issue the appropriate ddi_peek or ddi_poke call.

Note that the mmap() interface of the vmemem driver does not use the ddi_peek/ddi_poke calls. In such cases, access to all non-existent address space will cause a SIGBUS signal to be sent to the user program.

Devices that issue interrupts pose more of a problem when accessed from the user level. In some cases the devices can be programmed to not raise interrupts. Input/output in such cases may be done by making the user program 'busy-wait'. In other cases, the vmeintr driver can be used to receive the interrupt and signal a user program when the interrupt is raised. The user program can then look at the device registers and take appropriate actions.

4

VMEbus Interface ASIC: SCV64

This chapter contains an abbreviated version of the specifications for the Tundra SCV64 VME Controller Chip as they apply to the USP-1 design. The VMEbus to SBus Interface Controller (VSIC) is composed of both the SCV64 Controller and the KSIC Interface. Significant features involving VMEbus and SBus mapping and addressing are implemented in the KSIC interface and affect the operation and functionality of the SCV64 Controller.

4.1 Features

- VMEbus Rev. D compliant
- A32, A24:D32, D16, D08 (EO) Master capabilities and A64 capability when using DMA
- A16:D16, D08 master capabilities
- A32, A24 Slave capabilities with configurable slave window sizes of 8, 32, 64, or 128 Mbytes generating DVMA accesses
- Full VMEbus system controller functionality
- Automatic VMEbus SYSCON identification
- Interrupter and interrupt handling for all seven VMEbus interrupt levels
- VME slave access protection
- Multiple VMEbus request and release options
- Multiple VMEbus arbitration schemes
- Integrated DMA controller

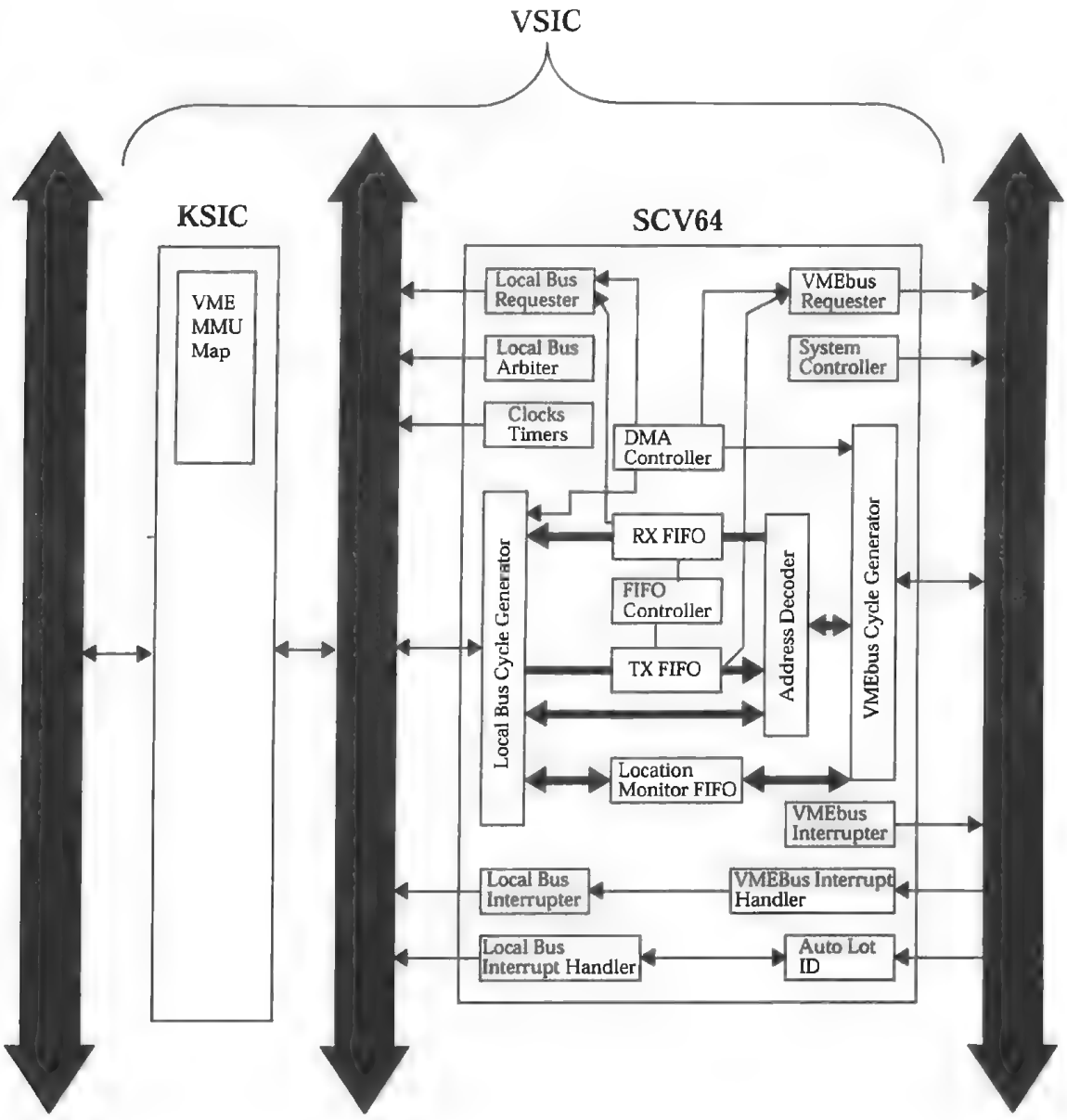


Figure 4-1. **VSIC (VMEbus to SBus Interface Component) Block Diagram**

4.2 General Description

The SCV64 is a single chip implementation of the 64-bit VMEbus interface compliant with the IEEE Rev. D. specification. The controller encompasses a rich set of programmable features and operational modes that make it ideally suited to CPU board implementations with both master and slave capabilities.

Caution — Not all features of the SCV64 Controller are applicable to the USP-1 design and certain capabilities and elements of the SCV64 have been superseded by custom implementations on the USP-1 SCV64 Registers

Table 4-1. **SCV64 Internal Register Map**

Address	Register	Name
0x1FF FD00 0000	DMA Local Address	DMALAR
0x1FF FD00 0004	DMA VMEbus Address	DMAVAR
0x1FF FD00 0008	DMA Transfer Count	DMATC
0x1FF FD00 000C	Control and Status	DCSR
0x1FF FD00 0010	VMEbus Slave Base Address	VMEBAR
0x1FF FD00 0014	Rx FIFO Data	RXDATA
0x1FF FD00 0018	Rx FIFO Address Register	RXADDR
0x1FF FD00 001C	Rx FIFO Control Register	RXCTL
0x1FF FD00 0020	VMEbus / VSB Bus Select	BUSSEL
0x1FF FD00 0024	VMEbus Interrupter Vector	IVECT
0x1FF FD00 0028	Access Protect Boundary	APBR
0x1FF FD00 002C	Tx FIFO Data Output Latch	TXDATA
0x1FF FD00 0030	Tx FIFO Address Output Latch	TXADDR
0x1FF FD00 0034	Tx FIFO AM Code and Control Bit Latch	TXCTL
0x1FF FD00 0038	Location Monitor FIFO Read Port	LMFIFO
0x1FF FD00 003C	SCV64 Mode Control	MODE
0x1FF FD00 0040	Slave A64 Base Address	SA64BAR
0x1FF FD00 0044	Master A64 Base Address	MA64BAR
0x1FF FD00 0048	Local Address Generator	LAG
0x1FF FD00 004C	DMA VMEbus Transfer Count	DMAVTC
0x1FF FD00 0050 - 007F	Reserved	
0x1FF FD00 0080	Status Register 0	STAT0
0x1FF FD00 0084	Status Register 1	STAT1
0x1FF FD00 0088	General Control Register	GENCTL
0x1FF FD00 008C	VMEbus Interrupter Requester Register	VINT
0x1FF FD00 0090	VMEbus Requester Register	VREQ
0x1FF FD00 0094	VMEbus Arbiter Register	VARB
0x1FF FD00 0098	ID Register	ID
0x1FF FD00 009C	Control and Status Register	CTL2
0x1FF FD00 00A0	Level 7 Interrupt Status Register	7IS
0x1FF FD00 00A4	Local Interrupt Status Register	LIS

Table 4-1. **SCV64 Internal Register Map** (Continued)

Address	Register	Name
0x1FF FD00 00A8	Level 7 Interrupt Enable Register	7IE
0x1FF FD00 00AC	Local Interrupt Enable Register	LIE
0x1FF FD00 00B0	VMEbus Interrupt Enable Register	VIE
0x1FF FD00 00B4	Local Interrupts 1 and 0 Control Register	IC10
0x1FF FD00 00B8	Local Interrupts 3 and 2 Control Register	IC32
0x1FF FD00 00BC	Local Interrupts 5 and 4 Control Register	IC54
0x1FF FD00 00C0	Miscellaneous Control Register	MISC
0x1FF FD00 00C4	Delay Line Control Register	DLCT
0x1FF FD00 00C8	Delay Line Status Register 1	DLST1
0x1FF FD00 00CC	Delay Line Status Register 2	DLST2
0x1FF FD00 00D0	Delay Line Status Register 3	DLST3
0x1FF FD00 00D4	Mailbox Register 0	MBOX0
0x1FF FD00 00D8	Mailbox Register 1	MBOX1
0x1FF FD00 00DC	Mailbox Register 2	MBOX2
0x1FF FD00 00E0	Mailbox Register 3	MBOX3
0x1FF FD00 00E4 - 01FC	Reserved	

4.3 VMEbus System Controller

4.3.1 Bus Arbitration

The arbiter is enabled whenever the USP-1 is configured as a System Controller. The SCV64 provides all arbitration modes defined in the VMEbus specification. The arbitration module provides four programmable arbitration modes:

- Full Round Robin
- Priority 3; Round Robin 2, 1, 0
- Priority 3, 2; Round Robin 1, 0
- Full Priority.

4.3.1.1 Arbitration Modes

All arbitration modes are determined by the ARB1 and ARB0 bits in the VARB register. Since the arbiter logic continuously samples the VARB register, any changes to these bits will immediately be reflected in the current arbitration mode. The default mode after reset is full Round Robin.

Table 4-2. **Arbitration Mode Settings**

Arbitration Mode	ARB1, ARB0 Setting
Full Round Robin	00
Priority 3; Round Robin 2, 1, 0	01
Priority 3, 2; Round Robin 1, 0	10
Full Priority	11

4.3.1.2 Arbitration Time-out

Arbitration time-out ensures that bus arbitration will resume if BBSY* is not asserted within 16 µs of a bus grant signal. The time-out returns the arbiter to its previous mode and outstanding requests. Arbitration time-out is enabled by default and is deactivated by clearing the ATEN bit in the VARB register.

4.3.2 Bus Timer

The SCV64 uses BERR* to terminate VMEbus data transfers if a slave does not respond within a specific programmed time. The time-out period before BERR* is driven low can be 16 µs, 32 µs, 64 µs (the default setting), or never, depending upon the value of the VXL1 and VXL0 bits in the VARB register. The Never setting should only be used during system debugging, since the VMEbus cannot recover from errors (such as incorrect addressing) in this mode. Note also that the Auto-ID function depends on a bus time-out.

The timer is held reset while DS1* and DS0* are high, and begins running when either data strobe is asserted. At time-out, the SCV64 asserts and maintains BERR* until both DS1* and DS0* are high. Note that a race condition can occur if a slave takes a long time to respond and asserts DTACK* coincidentally with the SCV64 assertion of BERR*. Under these conditions, the master may see DTACK* and BERR* simultaneously.

Table 4-3. **VMEbus Time-out Setting**

Time-out Period	VXL1, VXL0 Setting
Never	00
16 µs	01
32 µs	10
64 µs (default)	11

4.4 VME Slave Interface

The VME slave image memory map may be accessed from the VMEbus as either A32 or A24. Addressing mode A16 is not supported in slave mode. The base address for the A32 and A24 slave images (also referred to as slave windows) are programmed using the VMEBAR register. Details related to the programming of the VMEBAR register are provided below.

Table 4-4. **VMEbus Slave Base Address Register - VMEBAR (0x1FF FD00 0010)**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Not Used									A24SIZ		A24BA				
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Not Used							A32SIZ		A32BA						

Table 4-5. **VMEBAR Register Settings**

Name	State	Function
A24SIZ	0 1 2 3	512 Kbyte slave image 1 Mbyte slave image 2 Mbyte slave image 4 Mbyte slave image
A24BA	00 - 1F	Sets base address of A24 slave image[a] A24BA will be compared against VME A23:A19
A32SIZ -	xx	Sets size of A32 slave image
A32BA	xx	Sets base address of A32 slave image[b]

a. The A24 image base address can be programmed to any 512 Kbyte boundary or multiple of its programmed size, whichever is larger, within the 16 Mbyte A24 addressing space.

b. The A32 image base address can be programmed to any 128 Mbyte boundary and to any size from 4 KByte to 128 MByte.

Warning — The VSIC interface of the USP-1 design limits the slave address window to 8 Mbyte, 32 Mbyte, 64 Mbyte, or 128 Mbyte, configurable through jumpers.

4.5 VMEbus Interrupter

The SCV64 can generate VMEbus interrupts on any of the seven VMEbus interrupt levels. VMEbus interrupt level and interrupt generation are controlled by the VMEbus Interrupter Requester (VINT) Register.

Table 4-6. **VMEbus Interrupter Requester - VINT (0x1FF FD00 008C)**

Bit	Type	Function
7:4		Reserved
3	R	0 = No interrupt pending 1 = Interrupt is presently asserted

Table 4-6. **VMEbus Interrupter Requester - VINT (0x1FF FD00 008C)**

Bit	Type	Function
	W	0 = No effect 1 = Assert interrupt
2:0	R / W	000 = Interrupter clears itself 001 = VMEbus Interrupt Level 1 010 = VMEbus Interrupt Level 2 011 = VMEbus Interrupt Level 3 100 = VMEbus Interrupt Level 4 101 = VMEbus Interrupt Level 5 110 = VMEbus Interrupt Level 6 111 = VMEbus Interrupt Level 7

The corresponding interrupt vector is supplied upon IACK and programmed into the lower 8 bits of the VMEbus Interrupter Vector (IVECT) Register (0x1FF FD00 0024). Any reset will return the SCV64 registers to their default values. Thus all VMEbus interrupts will be deasserted.

4.6 VMEbus Interrupt Handling

Warning— The USP-1 implements custom VMEbus interrupt handling and acknowledgment. The following SCV64 material is presented for reference only

The SCV64 interrupt handler module accepts interrupt requests from three groups of sources listed according to their priority below:

1. Level 7 interrupts
2. General local interrupts
3. VMEbus interrupts.

All interrupts are channeled to a common interrupt handler and all interrupt acknowledge cycles are processed by a single acknowledgment module.

Each interrupt group has its own control register for enabling / disabling. The 7IE register controls level 7 interrupts, the LIE register controls general local interrupts and the VIE register enables VMEbus interrupts. An interrupt source is enabled by setting its corresponding control bit in its control register. Except the L7INMI, which is always enabled, all interrupts are reset by disabling and re-enabling through their respective control bits. The L7INMI is reset by clearing the NMICLR bit in the 7IE register.

Table 4-7. **Level 7 Interrupt Control Bits**

Level 7 Interrupts	Control Bits in 7IE Register 0x1FF FD00 00A8
*L7NMI	NMICLR
*L7IMEM	MEMIE

Table 4-7. **Level 7 Interrupt Control Bits**

Level 7 Interrupts	Control Bits in 7IE Register 0x1FF FD00 00A8
*L7IACF	ACFIE
BIMODE	BIE
*SYSFAIL	SYSFIE

Table 4-8. **Local Interrupt Control Bits**

General Local Interrupts	Control Bits in LIE Register 0x1FF FD00 00AC
LIRQ5	L5E
LIRQ4	L4E
LIRQ3	L3E
LIRQ2	L2E
LIRQ1	L1E
LIRQ0	L0E

Table 4-9. **VMEbus Interrupt Control Bits**

VMEbus Interrupts	Control Bits in VIE Register 0x1FF FD00 00B0
IRQ7	V7E
IRQ6	V6E
IRQ5	V5E
IRQ4	V4E
IRQ3	V3E
IRQ2	V2E
IRQ1	V1E

4.7 Location Monitor

The Location Monitor resides at the top longword (32 bits) of each A32 and A24 slave image and is accessible from both the local and VMEbus. The bottom (even) word of the Location Monitor is used for 16-bit messages. Since the Location Monitor replaces the longword of memory which otherwise exists at that address, writes to the Location Monitor are not turned into local cycles or entered into the RXFIFO. In addition, read accesses to the Location Monitor result in a bus error.

Writes to the Location Monitor are queued in the LMFIFO (32 bits wide and 31 stages deep) and read from the LMFIFO register. While there are entries in the LMFIFO, the SCV64 asserts LMINT and sets the LMHD bit in the DCSR register. Since LMINT is only asserted while there are entries in the LMFIFO, an interrupt service routine or polling of the LMINT signal can be used to test for entries in the LMFIFO and to support various Location Monitor functions.

The CPU may write to its own Location Monitor through its slave image as it would any address on the VMEbus. The SCV64 address decoder responds by redirecting the message to the LMFIFO. From there the message is handled in the same manner as a write from the VMEbus side. If the condition occurs in which the local CPU and VMEbus write to the Location Monitor at the same time, the race condition is resolved and the writes queued without loss of either of the message providing space is available in the LMFIFO.

One function of the Location Monitor is as a command to exit BI-mode. When there is a write to the Location Monitor, the SCV64 is removed from BI-mode and the BIMODE signal is cleared.

Table 4-10. **Location Monitor FIFO Read Port - LMFIFO (0x1FF FD00 0038)**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Location Monitor FIFO Output Stage - Byte 3								Location Monitor FIFO Output Stage - Byte 2							
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Location Monitor FIFO Output Stage - Byte 1								Location Monitor FIFO Output Stage - Byte 0							

4.8 DMA Transfers

The SCV64 performs all BLT and MBLT cycles through the DMA controller. During DMA transfers, the SCV64 is both the VME and local master. In case of a DMA write operation, the source address is contained in the DMA Local Address (DMALAR) Register.

Table 4-11. **DMA Local Address Register - DMALAR (0x1FF FD00 0000)**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Not Used					DLA - Byte 3			DLA (DMA Local Address) - Byte 2							
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DLA (DMA Local Address) - Byte 1								DLA (DMA Local Address) - Byte 0							0

DMA reads use the DMA VMEbus Address (DMAVAR) Register for the source address.

Table 4-12. **DMA VMEbus Address (DMAVAR) Register - (0x1FF FD00 0004)**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
DVA (DMA VME Address) - Byte 3								DVA (DMA VME Address) - Byte 2							
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DVA (DMA VME Address) - Byte 1								DVA (DMA VME Address) - Byte 0							

Data from the source address is transferred to the RXFIFO or TXFIFO for DMA read and write operations respectively.

The various addressing and data transfer modes applicable to the DMA operation are selected using the MODE register. The required bit settings for the supported addressing (A64, A32, A24) and data modes (D32, D16) are indicated below.

For A64 transfers, the MA64BAR register is used for the upper 32-bits of the source or destination address.

The size of the DMA transfer is controlled by both the DMA Transfer Count (DMATC) Register and the DTCSIZ bit in the MODE register. With the DTCSIZ bit cleared (default), the DMATC register will count up to 4,096 32-bit transfers (16 KBytes). If the DTCSIZ bit is set, the DMATC register will count up to 1,048,576 32-bit transfers (4 MBytes).

Table 4-13. **DMA Transfer Count Register - DMATC (0x1FF FD00 0008)**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Not Used								Not Used				DTC			
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DTC (DMA Transfer Count)								DTC (DMA Transfer Count)							

The transfer count for VMEbus cycles is recorded in the DMA VMEbus Transfer Count (DMAVTC) Register. The difference between the value in the DMATC register and the DMAVTC is equal to the number of entries in the RXFIFO or TXFIFO.

Table 4-14. **DMA VMEbus Transfer Count Register - DMAVTC (0x1FF FD00 004C)**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Not Used								Not Used				DMA VME Transfer Count			
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DMA VME Transfer Count								DMA VME Transfer Count							

5

Programmer's Visible State

5.1 Introduction

This chapter provides an overview of the address partitioning and visible software registers. The physical address associated with each of these registers is listed along with a description of the register.

Detailed explanations for the workings of the chips whose registers are described here can be found in the specifications for the USC, U2S, MACIO, and SLAVIO.

Warning — Registers designated as write-only may be read, but the data returned is undefined. No error is reported for such an access. Software should never rely on the value returned. Writes to read-only registers have no effect. No error is reported for this type of access.

5.2 Physical Address Space Allocation

The USC interfaces to three UPA ports and Main Memory. When a UPA master generates a request with an explicit address the System Controller decodes this address to determine its destination. The mapping between UPA ports and their respective addresses is hardwired in the System Controller in accordance with the Sun4u architecture guidelines. Refer to *Figure 5-1* below for the format of all I/O accesses. The top three bits are set to '1', the next 5 bits comprise the UPA port number (UPN) of the desired UPA port, and the lower 33 bits contain the port address. *Table 5-1* specifies the Physical Address to Port ID mapping in the System Controller. *Table 5-2* specifies the Physical Address to SBus mapping in the U2S.

Figure 5-1. **System Register Address Formation**

Table 5-1. **System Controller Address Map**

Address Range in UPA Address <40:0>[a]	Size	UPA Port Addressed	Port ID[b]	Notes
0x000 0000 0000 - 0x0FF FFFF FFFF	1 Terabyte	Main Memory	N/A	The System Controller responds to entire address range[c]
0x100 0000 0000 - 0x1BF FFFF FFFF	768 Gbytes	Reserved[d]	N/A	
0x1C0 0000 0000 - 0x1C1 FFFF FFFF	8 Gbytes	Processor 0	0	
0x1C2 0000 0000 - 0x1C3 FFFF FFFF	8 Gbytes	Reserved[d]	1	
0x1C4 0000 0000 - 0x1FB FFFF FFFF	224 Gbytes	Reserved[d]		
0x1FC 0000 0000 - 0x1FD FFFF FFFF	8 Gbytes	Reserved	30	
0x1FE 0000 0000 - 0x1FE FFFF FFFF	4 Gbytes	U2S	31	U2S Internal Register
0x1FF 0000 0000 - 0x1FF FFFF FFFF	4Gbytes	U2S	31	System I / O

a. Accesses to slave addresses **must** be non-cacheable.

b. Port ID is the Master ID (MID) for devices which support mastership.

c. On the USC, the maximum memory is 1 Gbyte. Addresses beyond 1 Gbyte will wrap.

d. Accesses to reserved regions will result in a time-out to the initiator.

Table 5-2. **U2S SBus Address Map**

Address Range in UPA Address <40:0>	Size	SBus Port Addressed	Notes
0x1FF 0000 0000 - 0x1FF 0FFF FFFF	256 Mbyte	Slot 0	
0x1FF 1000 0000 - 0x1FF 1FFF FFFF	256 Mbyte	Slot 1	
0x1FF 2000 0000 - 0x1FF 2FFF FFFF	256 Mbyte	Slot 2	VMEbus
0x1FF 3000 0000 - 0x1FF 3FFF FFFF	256 Mbyte	Slot 3	MACIO #2
0x1FF 4000 0000 - 0x1FF CFFF FFFF	2.256 Gbyte	Reserved	
0x1FF D000 0000 - 0x1FF DFFF FFFF	256 Mbyte	APC (Slot 13)	
0x1FF E000 0000 - 0x1FF EFFF FFFF	256 Mbyte	MACIO #1 (Slot 14)	
0x1FF F000 0000 - 0x1FF FFFF FFFF	256 Mbyte	SLAVIO (Slot 15)	

Note — Invalid accesses to valid regions, for example an NC access to memory, will generate a bus error. Valid accesses to reserved regions will generate a time-out error. Refer to Chapter 10, UPA Packet Handling for details of UPA packet handling. Transactions marked RTO will cause a time-out, while transactions marked error will report a bus error.

5.3 Memory Address Assignments

Main memory on the USP-1 spans a 1 Gbyte region starting at physical address 0x000 0000 0000. The USP-1 supports up to four 256 MByte Modules.

Addresses mapped to memory must be cacheable. Transfers between any UPA port and memory has to be done in cache line size of 64 bytes. Non-cacheable accesses to memory are not supported by Sun4u and will be treated as an error.

Each memory module in the system has a unique address range based on type and size of the memory module. Software can identify type and size of the memory module based on its address range. The address ranges for each memory module in are listed in *Table 5-3* below.

Table 5-3. **Memory Module Address Assignments (SBus Reference Platform)**

Memory Module Number	Memory Module Size	Address Range (PA [30:0])
0	128 Mbyte	0x0000.0000 - 0x07FF.FFFF
0	256Mbyte	0x0000.0000 - 0x0FFF.FFFF
1	128 MBytes	0x0000.0000 - 0x17FF.FFFF
1	256Mbyte	0x1000.0000 - 0x1FFF.FFFF
2	256Mbyte	0x2000.0000 - 0x2FFF.FFFF
3	256Mbyte	0x3000.0000 - 0x3FFF.FFFF

Note — Memory Modules #0 and #1 may be either 128 MBytes or 256 MBytes..

5.4 System Controller Registers

The System Controller internal address map is shown in Table 5-7. Note that the addresses specified are only 8 bits wide. This is because the addresses are not system addresses, but System Controller internal addresses. The System Controller is not addressed directly by software, rather the internal address map is addressed indirectly.

For all the registers that follow, not all 32 bits are specified. If a field is not specified explicitly, then it is considered RESERVED. Writes to RESERVED fields are ignored, and data read from a reserved field is always 0. Reads from an unimplemented System Controller internal addresses return UNDEFINED data. Writes to unimplemented System Controller internal addresses have unknown effects, as the address may wrap to an implemented address.

5.4.1 System Controller System Addresses

The System Controller requires two system addresses be allocated to it. The first system address is for the SC_Address Register. This need only be a single byte. The second system address is for the SC_Data Register. This must be a 32-bit word address, but it is accessed by software as four individual bytes.

Access to the SC_Address Register and the SC_Data Register is done through the SLAVIO generic port (EBus). Some address bits in the generic port address space are decoded to select the System Controller registers.

Table 5-4. **Physical Address of System Controller Registers**

Register	Physical Address
SC_Address Register	0x1FF F130 0000
SC_Data Register	0x1FF F130 0004

5.4.1.1 SC_Address Register

– **Address: 0x1FF F130 0000**

The SC_Address Register is a byte access register. *Table 5-5* lists the values software writes into this register in order to access other internal registers. The SC_Address Register can be read and written.

Table 5-5. **SC_Address Register Bit Definitions**

Field	Bits	Pupstate[a]	Description	Type
Address	7:0	X	Address pointer to System Controller internal register map	R / W

a. Power Up State

5.4.1.2 SC_Data Register

– **Address: 0x1FF F130 0004**

Software accesses the SC_Data Register with four consecutive, atomic byte access to addresses 0x1FF F130 0004, 0005, 0006, and 0007. Since atomicity of the four accesses is not guaranteed by hardware, software must guarantee this by the use of a mutex lock or other such mechanism. Since the access to this register is an indirect access to another register in the System Controller, software must take care to access the bytes in the appropriate order.

Note — Byte ordering within the SC_Data Register is big-endian (e.g., byte 0 corresponds to bits 31:24).

For read and write accesses, the order is byte 0, followed by byte 1, then 2, then 3. Reading byte 0 causes the entire word to transfer to a holding register for subsequent reading and returns byte 0 data; subsequent reads of bytes 1, 2, and 3 return the contents of the holding register. Writes are similar in that writes of bytes 0, 1, and 2 are stored in a temporary write register until byte 3 is written. Then the entire word is transferred to the requested internal register.

Table 5-6. **SC_Data Register Bit Definitions**

Field	Bits	Pupstate[a]	Description	Type
Data	31:0	X	Reading byte 0 initiates an indirect read Writing byte 3 initiates the indirect write	R / W

a. Power Up State

Table 5-9. **SC_Control Register Bit Definitions**

Field	Bits	Pupstate	Description	Type
POR	31	*[a]	Power On Reset - Asserted only after the System Controller sees the assertion of Sys_Reset_L	R / W1C
SOFT_POR	30	*	Set if the last reset was due to software setting the Soft_Power_On Reset Bit	R / W
SOFT_XIR	29	*	Set by software to indicate an XIR Reset Condition	R / W
B_POR	28	*	Set if the last reset was due to the assertion of P_Reset_L on the System Controller. This bit can be set due to a scan reset or through a reset generated when writing to the frequency margining register.	R / W1C
B_XIR	27	*	Set if the last reset was due to the assertion of an X_Reset_L on the System Controller. This bit can also be set by a scan.	R / W1C
WAKEUP	26	*	Set if the last reset was due to a wakeup event. This bit only affects port 0 on SBus Reference Platform.	R / W1C
FATAL	25	*	Set if a FATAL error was detected by the System Controller or reported to it. When this bit is set, a system reset will occur (System Controller CSR bits are not affected).	R / W1C
RESERVED	24	0	Reserved	R0
IAP	23	0	Invert Parity on the address busses	R / W
EN_WKUP_POR	22	0	Enable wakeup POR generation. Cleared by POR, SOFT_POR, B_POR, FATAL, and WAKEUP	R / W
RESERVED	21:0	0	Reserved	R0

a. The highest priority reset source will have its bit set. Only the bits marked with '*' will be set.

- **POR - Power On Reset**

  This bit is set if the last reset of the System Controller was due to the assertion of SYS_RESET_L by an external source, and will occur whenever the machine power cycles.

- **SOFT_POR - Soft Power On Reset**

  Writing a 1 to this bit has the same effect as power-on reset, except a different status bit in the System Controller Control Register is set. Memory refresh is not affected. Writing a 0 to this bit clears it and has no other effect.

- **SOFT_XTR - Soft Externally Initiated Reset**

  Writing a 1 to this bit causes the System Controller to assert UPA_XIR_L for 1 cycle. This should cause software trap for any UPA that is a processor.

- **B_POR - Button Reset**

  This bit is set as a result of a "button" reset which is caused by an external switch that asserts P_RESET_L on the System Controller. It can also be set by scan and the frequency margining reset. Memory refresh is not affected. The actions and results of this reset are identical to that of Power-on Reset, except a different status bit is set.

- **B_XIR - XIR Button Reset**

  This bit is set as a result of a "button" XIR Reset which is caused by an external switch that asserts the X_RESET_L on the System Controller, or by scan. The actions and results of this reset are identical to that of SOFT_XIR, except a different status bit is set.

- **WAKEUP - Wakeup**

  This bit is set if a wakeup event occurred from one of the UPA Ports and EN_WKUP_POR is true. A wakeup event is an interrupt to a port that has its 'Sleep' Bit set - See SC_Port_Config Register Definitions. Upon a wakeup event, the System Controller asserts UPA_RESET_L<0> to all nodes that can sleep. Software reads this bit to determine that the last reset it received was due to a wakeup. This allows it to enter the wakeup related code.

- **FATAL**

  This bit is set if one of the following fatal errors was detected by the System Controller:

  1. Address Bus Parity Error
  2. P_FERR Reported by a UPA
  3. A Master Port overflowed its queue
  4. Handshake error between DPS and Memory Controller.

  Software should also read the registers listed above to determine the actual source of the error and clear any associated bits.

Note — In the USC, only cases 1, 3, and 4 will generate a fatal error.

- **IAP - Invert Address Parity**

  This bit is used by diagnostic software to invert the address parity generated by the System Controller. This is useful for checking the hardware and software involved in logging address parity errors. It is cleared after any POR and unchanged after XIR, or wakeup.

- **E_WKUP_POR - Enable Wakeup POR**

  Enables POR generation upon receipt of a wakeup event directed to a port with the 'S_Sleep' bit set. Software should set this bit when putting the system to sleep. This bit is cleared on power up, including a wakeup power up.

Note — Interrupts to ports which don't have the 'S_Sleep' bit set, including interrupts to invalid ports, will not cause a wakeup reset.

Table 5-10. **System Controller Reset Strategy**

Reset Type	Bit Set	Signal Assertions	SC FSM's	System Controller Status Registers
Power On	POR	UPA_Reset_L <1:0> for 200 msec after power is valid.	All Reset	All Reset
Software	SOFT_POR	UPA_Reset_L <1:0> for 6.6 msec	All Reset	All reset except SOFT_POR
XIR	SOFT_XIR	XIR to all processors	Unaffected	Only XIR source is affected
Button Reset	B_POR	UPA_Reset_L <1:0> for 6.6 msec	All Reset	All reset except B_POR
XIR Button	B_XIR	UPA_XIR_L for 1 System Cycle	Not Affected	Only B_XIR source is affected
Wakeup Interrupt	WAKEUP	UPA_Reset_L <0>	System Controller's UPA Arbiter is Reset	Only WAKEUP source is affected
Fatal Error Condition	FATAL	UPA_Reset_L for 6.6 msec	All Reset	Only FATAL source is affected

Among the reset bits, only one will be set when reset terminates. If multiple resets occur simultaneously, the following order will be chosen for the reporting:

1. POR
2. FATAL
3. B_POR
4. WAKEUP
5. SOFT_POR
6. B_XIR
7. SOFT_XIR.

5.4.3.2 SC_ID Register

— **Address: 0x04**

This register contains read-only ID information for the System Controller.

Table 5-11. **SC_ID Register Description**

Field	Bits	PUP State	Description	Type
JEDEC	31:16	0x3340	System Controller's JEDEC ID; 0x3340 is for the USC	R
UPANUM	15:12	0x3	The number of UPA ports supported by this implementation of the System Controller	R
RESERVED	11:8	0	Reserved	R0

Table 5-11. **SC_ID Register Description**

Field	Bits	PUP State	Description	Type
IMPLNUM	7:4	0	Implementation Number	R
VERNUM	3:0	0	Version Number; Starts at 0 and increments for each revision of the ASIC	R

5.4.3.3 SC_Perf0 Register

– **Address: 0x08**

This is a 32-bit read-only register. The value read back contains the counts of the event selected by the SC_PerfCtrl Register. Whenever this register is read, the SC_PerfShadow Register is updated with the contents of the SC_Perf0 counter. When the counter reaches its maximum count, it will wrap around to 0x0 and continue to count. Software needs to detect and handle the over-flow condition. A list of possible selections for counter 0 is shown in *Table 5-13* below.

Table 5-12. **SC_Perf0 Register Bit Definitions**

Field	Bits	Description	Type
CNT0	31:0	Contains value for event counter 0	R

Table 5-13. **SC_Perf0 Register Event Sources**

SEL	Event Sources
0x0	System Clock Count
0x1	Number of P_Requests from all sources
0x2	Number of P_Requests from Processor 0
0x3	Not Used.
0x4	Number of P_Requests from U2S
0x5	Number of cycles the UPA 128-bit data bus is busy
0x6	Number of cycles the UPA 64-bit data bus is busy
0x7	Number of cycles stalled during PIO
0x8	Number of Memory Requests issued
0x9	Number of cycles Memory Controller is busy[a]
0xA	Number of cycles stalled because of a Pending Transaction Scoreboard hit
0xB	Number of Coherent Write Miss Requests from P0 (RDO)
0xC	Number of Coherent Write Miss Requests from P1 (RDO)
0xD	Number of Coherent Intervention Transactions (CPI + INV + CPB + CPD)
0xE	Number of Data Transactions (RDO + RDD + WRB + WRI) from U2S
0xF	Number of Coherent Read Transactions (CPB + CPD) issued

a. This includes all events which cause the memory controller to be non-idle including memory references and refresh.

5.4.3.4 SC_Perf1 Register

– **Address: 0x0C**

This is a 32-bit read-only register. The value read back contains the counts of the event selected by the SC_PerfCtrl Register. Whenever this register is read, the SC_PerfShadow Register is updated with the contents of the SC_Perf1 counter. When the counter reaches its maximum count, it will wrap around to 0x0 and continues counting. Software needs to detect and handle the overflow condition. A list of possible selections for counter 1 is shown in the *Table 5-15* below. Note that this list contains some duplications with the SC_Perf0 Register (*Table 5-13*), and some unique items. Common sources retain the same SEL number.

Table 5-14. **SC_Perf1 Register Bit Definitions**

Field	Bits	Description	Type
CNT1	31:0	Contains value for event counter 1	R

Table 5-15. **SC_Perf1 Register Event Sources**

SEL	Event Sources
0x0	System Clock Count
0x1	Number of P_Requests from all sources
0x2	Number of P_Requests from Processor 0
0x3	Not Used.
0x4	Number of P_Requests from U2S
0x5	Number of Read Requests from P0
0x6	Number of Coherent Read Misses from P0
0x7	Number of PIO accesses (NCxx) from P0
0x8	Number of Memory Requests issued
0x9	Number of Memory Requests completed
0xA	Number of Read Requests from P1
0xB	Number of Coherent Read Misses from P1
0xC	Number of PIO accesses (NCxx) from P1
0xD	Number of Coherent Write Transactions (CPI + INV) issued
0xE	Number of Data Transactions (RDO + RDD + WRB + WRI) from U2S
0xF	Reserved

5.4.3.5 SC_PerfShadow Register

– **Address: 0x10**

This is a 32-bit read-only register. The value read back contains the count of the event shadowed by the read of the last SC_Perf{0,1} Register. This register provides a mechanism to 'atomically' read the values of both counters. The SC_PerfShadow Register is sampled at the time byte 0 is read for either performance register.

Table 5-16. **SC_PerfShadow Register Bit Definitions**

Field	Bits	Description	Type
CNT	31:0	Contains value for event counter 0 or 1	R

5.4.3.6 SC_PerfCtrl Register

– **Address: 0x20**

The SC_PerfCtrl Register controls the events to be monitored by the SC_Perf{0,1} Registers. The event counter in the SC_Perf{0,1} Register will be reset when the 'CLR{0,1}' bit is asserted.

Note — There are only two counters with each counter multiplexing one of sixteen possible sources. It is not possible to track more than 2 events simultaneously.

Table 5-17. **SC_PerfCtrl Register Bit Definitions**

Field	Bits	Description	Type
RESERVED	31:16	Reserved	R0
CLR1	15	Clears the counter 1	W
RESERVED	14:12	Reserved	R0
SEL1	11:8	Select event source for counter 1. Counter 1 is cleared when CLR1 field is written and resets to 0xF.	R / W
CLR0	7	Clears the counter 0	W
RESERVED	6:4	Reserved, Read as 0	R0
SEL0	3:0	Select event source for counter 0. Counter 0 is cleared when CLR0 field is written and resets to 0xF.	R / W

5.4.3.7 SC_Debug_Pin_Ctrl Register

– **Address: 0x30**

The SC_Debug_Pin_Ctrl Register controls the internal state which is displayed on the debug pins for the USC. Four pins are provided on the USC to make internal events visible. *These pins are for hardware debug only.*

Table 5-18. **SC_Debug_Pin_Crtl Register Bit Definitions**

Field	Bits	Description	Type
RESERVED	31:16	Reserved	R0
SOURCE	15:0	Selects the source for debug information	R / W

Note — Values written to this register should have at most one bit in the word set. If more than one bit is set, then the result is undefined.

Table 5-19. **SC_Debug Pin Definitions**

SC_Debug Value	Internal Signal	Description
0x0 (default)	BB_DbgReset	POR, PB_Wakeup, BX_HardReset, BX_SoftReset
0x1	BB_DbgEBusIf	EBusIfState[3:0]
0x2	BB_DbgResetFsm	rsIDLE, rsSTRESET, rsRESET, ResetDone
0x4	PB_DbgNC0	ReqVal0, SlaveGrt0, SlaveOutVal, PD_ErrReg0[Val]
0x8	PB_DbgNC1	ReqVal1, SlaveGrt1, SlaveOutVal, PD_ErrReq1[Val]
0x10	PB_DbgNC2	ReqVal2, SlaveGrt2, SlaveOutVal, PB_Wakeup
0x20	PB_DbgC0	SBReq0, SBGrt0, RMW, PM_ReqVal
0x40	PB_DbgC1	SBReq1, SBGrt2, RMW, PM_ReqVal
0x80	PB_DbgC2	SBReq2, SBGrt2, WBCancel, PD_CohReq[Val]
0x100	PB_DbgArb	State[1:0], CurrentMaster[1:0]
- 0x200	MB_DbgMD	MD_Req, MD_DataAck0, MD_DataAck1, DM_ReqAck
0x400	MB_DbgPM	PM_ReqVal, MP_QFull, MB_DbgReadBusy, MB_DbgWriteBusy
0x800	MB_DbgInt	MB_DbgRefReq, MB_DbgRefReqDone, MB_DbgRefBusy, MB_DbgMCBlocked
0x1000	DB_DbgArb	SchdReq_m, SchdReq_f/u/e, SchdGrnt_m, SchdGrnt_f/u/e
0x2000	DB_DbgMem	MD_Req, MD_SnpStat, MD_SnpDone, SchdGrnt
0x4000	DB_DbgFfb	FfbWr[Val], ScheduleWrs, ValReqs, SchdReq

5.4.4 System Controller UPA Port Interface Registers

There are multiple copies of these registers (as specified). They are used to enable / disable certain port features. Each of the registers is listed individually to indicate which bits are hardwired in this implementation. Those bits which are hardwired (read only) are shown with the normal field name rather than being listed as reserved.

Implementations must guarantee that software setting of queue sizes beyond the implemented maximum are benign. The suggested behavior is to default any value greater than maximum to the maximum value. In other words, if a queue size field is 4 bits, an implementation could choose to limit the maximum value to 9. Programming a value larger than this in the field must not cause failure. Implemented values less than the field maximum should be indicated.

5.4.4.1 P{0,1}_Config Register

– **Address: 0x40 and 0x48**

The USP-1 has two Port Configuration registers.

Table 5-21. **P0_Status and P1_Status Register Bit Definitions**

Field	Bits	Pupstate	Description	Type
IADDR	30	0	The IADDR bit is set if the System Controller detects an attempt to send a request (read or write) to an illegal address or a nonexistent port. Accesses forwarded to valid ports will not set this bit regardless of the reply. Writes to processor ports also set IADDR. This is an advisory bit. Refer to Section 9.2.2, Non-Fatal Hardware Errors for more information.	R / W1C
IPORT	29	0	The IPORT bit is set if an attempt to send an interrupt packet to an illegal destination port is made, or if the destination port's SPIQS field is set to 0. This is an advisory bit. The interrupt packet is dropped (normal ack). Refer to Section 9.2.2, Non-Fatal Hardware Errors.	R / W1C
IPRTY	28	0	The IPRTY bit is set if an attempt to send an interrupt packet to an illegal destination port is made, or if the destination port's SPIQS field is set to 0. This is an advisory bit. The interrupt packet is dropped (normal ack). Refer to Section 9.2.2, Non-Fatal Hardware Errors.	R / W1C
MC0OF	27	0	Master Class 0 Overflow. Set if the master tried to send a packet when no space was available in the queue. This condition causes a chip reset. Refer to Section 9.2.1, Fatal Hardware Errors.	R / W1C
MC1OF	26	0	Master Class 1 Overflow. Set if the master tried to send a packet when no space was available in the queue. This condition causes a chip reset. Refer to Section 9.2.2, Fatal Hardware Errors.	R / W1C
MC0Q	25:23	See *Table 5-22*	These bits indicate to system software the number of requests packets that can be issued to master class 0 before its queue overflows.	R
MC1Q	22:20	See *Table 5-22*	These bits indicate to system software the number of request packets that can be issued to master class 1 before its queue overflows.	R
MC1Q	19	0	Returns 0 when read.	R
RESERVED	18:0	0	Reserved	R0

a. Refer to the FATAL bit in SC_Control Register.

Table 5-22 below gives the Port Number and the number of 2 cycle requests for that master's input queues, as specified by the 'MC1Q' and 'MC0Q' bits in the P{0,1}_Status Register.

Table 5-22. **P{0,1}_Status Register MC0Q and MC1Q Initialization Values**

Port / Class	Type of Master	Queue Size
Port 0 / Class 0	Processor 0	1

Table 5-22. **P{0,1}_Status Register MC0Q and MC1Q Initialization Values**

Port / Class	Type of Master	Queue Size
Port 0 / Class 1	Processor 0	4
Port 2 / Class 0	U2S[a]	2

a. There is only 1 class is allowed for the U2S.

5.4.4.3 SYSIO_Config Register

– **Address: 0x50**

Table 5-23. **SYSIO_Config Register Bit Definitions**

Field	Bits	Pupstate	Description	Type
MD	31	1	Master Disable. If set, the System Controller will block all requests in its master request queue for this master port. Once enabled, any requests which exist in the master request queues will proceed. Must be set to 0 for DVMA or interrupts.	R / W
RESERVED	30:28	0	Reserved; would be Slave Sleep for a port implementing this bit.	R
SPRQS[a]	27:24	1	Slave P_request queue size. Software initializes this field with the size of 2 Cycle Packets in the corresponding slave request queue.	R / W
SPDQS	23:18	4	Slave Port Data Queue Size. Software initializes this field with the length in address packets of the corresponding slave data queue[b].	R / W
RESERVED	17:16	0	Reserved; would be Slave Port Interrupt Queue Size for devices implementing this feature.	R
SQUEN	15	0	Software sets this bit when updating SPRQS, SPDQS and SPIQS. These fields must be updated simultaneously.	W
Oneread	14	1	U2S may be set as a oneread or a multiread device. Value should be programmed after a probe of the UPA Port ID register for the U2S.	R / W
RESERVED	13:0	0	Reserved.	R

a. An implementation may choose to set a maximum value which is less than what can be programmed in this register. In that case, a larger value must still result in correct operation, but not the desired number of requests being forwarded.

b. The UPA specification dictates that this field be present, but hardware ignores any nonzero value. If SPDQS is nonzero, then hardware assumes its value to be 4 times SPRQS.

5.4.4.4 SYSIO_Status Register

– **Address: 0x54**

This register is present for architectural consistency. No mastership is supported by the FFB, and all fields are reserved and will read back as 0.

Table 5-24. **SYSIO_Status Register Bit Definitions**

Field	Bits	Pupstate	Description	Type
FATAL[a]	31	0	The FATAL bit is set when the port sends a P_FERR error reply or when a P_RASB reply is made when Oneread is FALSE. This condition will cause a chip reset. Refer to Section 9.2.1, Fatal Hardware Errors.	R / W1C
IADDR	30	0	The IADDR bit is set when the port attempts to send a request (read or write) to an illegal address or a nonexistent port. This is an advisory bit. Refer to Section 9.2 2, Non-Fatal Hardware Errors.	R / W1C
IPORT	29	0	The IPORT bit is set when the port attempts to send an interrupt packet to an illegal destination port. This is an advisory bit. Refer to Section 9.2.2, Non-Fatal Hardware Errors.	R / W1C
IPRTY	28	0	The IPRTY bit is set when a packet sent from the port resulted in the System Controller detecting an address parity error. This condition can cause a chip reset. Refer to Section 9.2.1, Fatal Hardware Errors.	R / W1C
MC0OF	27	0	Master Class 0 Overflow. Set when the master attempts to send a packet when no space was available in the queue. This condition causes a chip reset. Refer to Section 9.2.1, Fatal Hardware Errors.	R / W1C
RESERVED	26	0	Reserved; would be Master Class1 Overflow for ports implementing this feature.	R0
MC0Q	25:23	See *Table 5-22*	These bits indicate to system software the number of requests packets that can be issued to Master class 0 before its queue overflows.	R
RESERVED	22:20	0	Reserved; would indicate Master Class 1 request packets for ports implementing this feature.	R
RESERVED	19:0	0	Reserved	R0

a. Refer to the FATAL bit in SC_Control Register.

5.4.5 System Controller Memory Controller Registers

The registers shown below control the functionality of the System Controller Memory Controller(s). The SC-UP is designed to be the most efficient, high performance, single bank memory controller possible.

Initialization of the Mem_Control{0,1} Registers should be performed in accordance with the probing algorithm.

Mem_Control0 Register

- **Address: 0x60**

The Mem_Control0 Register contains a number of bits which program the Refresh Controller in the Memory Controller.

Table 5-25. **Mem_Control0 Register Bit Definitions**

Field	Bits	PupState	Description	Type
RefEnable	31	0	Refresh enable	R / W
RESERVED	30:12	0	Reserved	R0
Modules Present	11:8	0xF	Determines which Memory Modules to refresh.	R / W
RefInterval	7:0	0x00	Interval Between refreshes. Each encoding is 8 system clocks.	R / W

- **RefEnable**

Main memory is composed of dynamic RAMs, which require periodic refreshing in order to maintain the contents of the memory cells. The RefEnable bit is used to enable refresh of main memory.

- 0 = disable refresh
- 1 = enable refresh

Disabling refresh causes the RefInterval timer to stop counting. POR is the only condition to clear RefEnable. SOFT_POR, B_POR, FATAL, WAKEUP, B_XIR, AND SOFT_XIR leave RefEnable unchanged.

Note — This bit may be written to at any time. Any refresh operation in progress at the time of clearing will be aborted with a possible loss of data.

- **MemModPresent <3:0>**

This field is used to indicate the presence / absence of memory modules so the System Controller does not spend more time than is necessary refreshing unpopulated memory modules. A zero in this field indicates the corresponding memory module is not present, a 1 indicates presence. These bits must be set by software after probing. The memory module to bit correspondence is given in *Table 5-26*.

Table 5-26. **MemModPresent Encoding Values**

MemModPresent	Memory Module Number
0	0
1	1
2	2
3	3

Note — Refresh must be disabled first by clearing the RefEnable bit before changing this field, or the RefInterval. Refresh may be enabled again simultaneously with writing MemModPresent and RefInterval. Failure to follow this rule may result in unpredictable behavior.

- **RefInterval**

RefInterval specifies the interval time between refreshes, in quanta of eight system cycles. Software should program RefInterval according to *Table 5-27*. Values given are in decimal and are derived from the following formula:

$$refValue = \frac{(Refresh\_Period)}{(Number\_Of\_Rows \times UPA\_Clock\_Period \times 8 \times Number\_Of\_Pairs)}$$

Note — One Themis Memory Module is the equivalent of a SIMM pair in the SBus Reference Platform.

Table 5-27. **RefInterval for Memory Modules**

Memory Module Pairs	UPA Cycle Time						
	10ns	12 ns	14 ns	15 ns	16 ns	18 ns	20 ns
1	195	162	139	130	121	108	97
2	97	81	69	65	60	54	48
3	65	54	46	43	40	36	32
4	48	40	34	32	30	27	34

Note — A value of 0 corresponds to refresh disabled. Values of 1 and 2 are illegal and should not be used. They are not checked in the hardware.

5.4.5.1 Mem_Control1 Register

- **Address: 0x64**

The Mem_Control1 Register contains fields which control the read, write, and refresh timing for the DRAM memory modules. They allow software to optimize memory access timing for a particular system frequency.

The contents of Mem_Control1 Register may be changed as required by any electrical tuning of memory timing based on detailed Spice analysis. Please see *Table 5-30 Mem_Control1 Values as a Function of System Operating Frequency* on page 5-21 for the proper programming values of this register.

Note — 50/60 ns DRAMs are supported in the USP-1.

Table 5-28. **Mem_Control1 Register Bit Definitions**

Field	Bits	PupState	Description	Type
RESERVED	31:13	0	Reserved	R0
CSR	12	1	CAS* to RAS* Delay for CBR refresh cycles	R / W
WPC1	11:10	11	Page Cycle 1 write	R / W
RCD	9	1	RAS* to CAS* Delay	R / W
CP	8	1	CAS* Precharge	R / W

Table 5-28. **Mem_Control1 Register Bit Definitions** (Continued)

Field	Bits	PupState	Description	Type
RP	7:6	11	RAS* Precharge	R / W
RAS	5:4	11	Length of RAS* for Precharge	R / W
PC0	3:2	11	Page Cycle 0	R / W
PC1	1:0	11	Page Cycle 1	R / W

Note — The fields of this register must be initialized before ANY memory operation, including refresh, can begin. Should software wish to change any of these values, it must first inhibit all memory references. Then, it must disable refresh. The control processor must wait a few dozen clocks after disabling refresh to insure that any pending refresh operation has completed. Accesses on the EBus may be sufficiently slow to satisfy this requirement. New values may now be programmed.

- **CSR - CAS before RAS Delay Timing**

  Controls the CAS* to RAS* assertion delay for CAS* before RAS*(CBR) refresh cycles.

  – 0 = 1 cycle between CAS* and RAS*

  – 1 = 2 cycles between CAS* and RAS*

- **WPC1 - Page Cycle 1 Write Timing**

  Controls the RAS* and CAS* low time during writes of the second block of data.

  – 00 = RAS* and CAS* low for 2 system clocks

  – 01 = RAS* and CAS* low for 3 system clocks

  – 10 = Reserved

  – 11 = Reserved

- **RCD - RAS* to CAS* Delay**

  RCD controls the RAS* to CAS* delay during the initial part of the read or write memory cycle.

  – 0 = 2 system clocks between the assertion of RAS* and the assertion of CAS*

  – 1 = 3 system clocks between the assertion of RAS* and the assertion of CAS*

- **CP - CAS* Precharge**

  CP controls the CAS* Precharge time in between page cycles.

  – 0 = 1 clock of CAS* Precharge

  – 1 = 2 clocks of CAS* Precharge

- **RP - RAS* Precharge**

  RP controls the RAS* Precharge time during reads and writes.

  – 00 = 3 System clocks of RAS* Precharge

  – 01 = 4 System clocks of RAS* Precharge

- 10 = 5 System clocks of RAS* Precharge
- 11 = Reserved

- **RAS***

RAS is used to control the length of time that RAS* is asserted during refresh cycles.

- 00 = RAS* asserted for 4 clocks
- 01 = RAS* asserted for 5 clocks
- 10 = RAS* asserted for 6 clocks
- 11 = Reserved

- **PC0 - Page Cycle 0**

PC0 controls the time to deassertion of CAS* for the read of the first half of the memory block. It does not affect the timing for writes, which is fixed at 4 clocks.

- 00 = CAS* asserted for 2 clocks
- 01 = CAS* asserted for 3 clocks
- 10 = CAS* asserted for 4 clocks
- 11 = CAS* asserted for 5 clocks

- **PC1 - Page Cycle 1**

PC1 Controls the timing of the RAS* and CAS* for the read of the second half of the memory block.

Table 5-29. **PC1 Field Timing Effects**

PC1		Relative Timing					
00 -	RAS*	L	L	L	H		
	CAS*	H	L	L	H		
01 -	RAS*	L	L	L	L	H	
	CAS*	H	L	L	L	H	
10 -	RAS*	L	L	L	L	H	H
	CAS*	H	L	L	L	L	H
11 -	RAS*	L	L	L	L	H	H
	CAS*	L	L	L	L	L	H

The values of this control register, as a function of system operating frequency, are given in *Table 5-30.*

Table 5-30. **Mem_Control1 Values as a Function of System Operating Frequency**

Clock Period (ns)	CSR	WPC1	RCD	CP	RP	RAS	PC0	PC1
10	0	01	1	0	10	10	11	11
12	0	00[a]	1	0	01	10	10	10

Table 5-30. **Mem_Control1 Values as a Function of System Operating Frequency** (Continued)

Clock Period (ns)	CSR	WPC1	RCD	CP	RP	RAS	PC0	PC1
14	0	00	1	0	00	01	01	01
15	0	00	0	0	00	01	10	01
16	0	00	0	0	00	00	10	01
18	0	00	0	0	00	00	01	01
20	0	00	0	0	00	00	01	00
Default	1	11	1	1	11	11	11	11

a. Timing for 12 ns cycle has not yet been confirmed. This value may go to 01 if the timing is not correct for this frequency.

5.5 I/O System

5.5.1 U2S (SYSIO) Registers

Warning — Register accesses to the U2S can be in any size from one byte up to 8 bytes. Sizes and locations for the registers are given in the sections which follow. Reads up to 8 bytes of any register are supported. Writes of any size up to 8 bytes are also supported. Writes MAY corrupt unwritten bits in the register (i.e, writes may result in all 8 bytes being written regardless of the indicated write size). Software must insure that only the proper sized accesses are used. No hardware checking is performed. Burst access to U2S registers is not permitted and will result in an error return for reads, and silent failure for writes. Misaligned access due to not setting the 'E' bit correctly in the TTE also yield unpredictable results.

Figure 5-2 below shows the rough breakdown of addresses within U2S. The upper 4 GByte are used for pass through, while the lower 4 GByte address internal registers to the U2S.

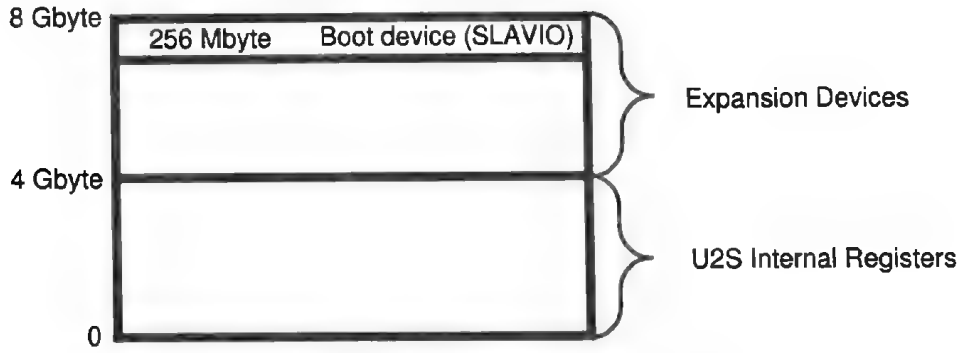


Figure 5-2. **Breakdown of U2S Address Space**

5.5.1.1 SYSIO UPA Port / ID Register

- **Address: 0x1FE 0000 0000**
- **Access Size: 8 bytes**

This read-only register includes information about identification and capability of the U2S UPA interface.

Table 5-31. **UPA Port ID Register Bit Definitions**

Field	Bits	Description	Type
FCODE_ESCAPE	63:56	Value is 0xFC, which is the FCODE for FERR. Software attempts to read this register as FCODE result in errors.	R
RESERVED	55:35	Reserved	R0
ECCNotValid	34	Indicates whether this port can generate ECC when sourcing data. Set to 0.	R
Oneread	33	Set if the slave port can allow only one outstanding slave read P_REQ transaction at a time. This bit is set to 0.	R
RESERVED	32:31	Reserved - Would encode PINT_RDQ for ports implementing this feature.	R0
PREQ_DQ	30:25	Specify the size of data queue in 16-byte unit. This field is 0x8 for U2S which has 128 bytes of data queue.	R
PREQ_RQ	24:21	Specify the size of PREQ_RQ queue. U2S can support 2 pending PREQ, this field is 0x2.	R
UPACAP	20:16	Bit 20: Set if the port can service interrupt; set to 0. Bit 19: Set if the port can generate interrupt; set to 1. Bit 18: Set if the port uses UPA_Slave_Int_L signal; set to 0. Bit 17: Set if the port has a cache; set to 0. Bit 16: Set if the port has master capability; set to 1.	R
JEDEC	15:0	JEDEC identification (0xEF07)	R

5.5.1.2 SYSIO_UPA_Config Register

- **Address: 0x1FE 0000 0008**
- **Access Size: 8 bytes**

This register indicates the queue sizes for each class of UPA request. Please refer to Uniprocessor System Controller User's Guide for the description of classes. The U2S only uses one request class for its transfer. The depth of queue supported by the System Controller for the U2S is two, therefore the 'SCIQ0' field should be programmed with 0x2. The initial value after reset is 0x1.

Table 5-32. **UPA Configuration Register Bit Definitions**

Field	Bits	Description	Type
RESERVED	63:8	Reserved	R0
SCIQ1	7:4	Unused - Read as 0 and write has no effect.	R0
SCIQ0	3:0	Size of input request queue for one master class in the System Controller. Software should set it to 0x2 during initialization.[a]	R / W

a. Software must insure that NO DMA is taking place when this field s changed. Once set, the value can not be reduced; it can only be increased.

5.5.1.3 U2S Control / Status Register

The U2S Control / Status Register provides the implementation and revision number of the U2S, enable clock output, and specifies the relative speed of the UPA and U2S clocks. It also enables address parity checking, inverts UPA address parity, and maintains the UPA MID set-up.

- **Address: 0x1FE 0000 0010**
- **Access Size: 8 bytes**

Table 5-33. **U2S Control / Status Register Bit Definitions**

Field	Bits	Description	Type
IMPL	63:60	Implementation number of U2S	R
VER	59:56	Revision number of this implementation	R
MID	55:51	UPA MID for U2S. Bits contain 0x1F on reset. Software should set up MID before allowing U2S to generate interrupt or DVMA. Refer to the Port ID listed in *Table 5-1* on page 5-2.	R / W
IGN	50:46	Interrupt Group Number; sets the five bits of the IGN field supplied in the first word of an interrupt packet. See *Table 5-67 Physical Address of Interrupt Mapping Registers* on page 5-42	R / W
RESERVED	45:5	Reserved	R0
CLK_OUT_EN	4	Clock output enable. Defaults to 1 (the clock output is enabled). Should be set to 0 by initialization software. Useful for checking the behavior of the internal PLLs.	R / W
APCKEN	3	Address Parity Check Enable - Defaults to 0 at power up. When set, any parity error detected results in a P_FERR reply. If clear, parity errors are still logged (in APERR), but the transaction continues as though the parity is correct.	R / W
APERR	2	Incoming system address parity error - Persistent across reset. Is set regardless of the value in APCKEN.	R / W1C
IAP	1	Invert UPA Address Parity - Reset to 0. U2S generates odd parity when this bit is set to 0 and even parity when set to 1.	R / W
MODE	0	Specify the speed of U2S clock relative to UPA clock. MODE = 1 if UPA clock is faster than U2S clock, and 0 if equal to or slower. Set to 0 upon reset.	R / W

Design of the U2S is optimized with the assumption that the UPA is running faster than the U2S clock. This assumption is true in normal operation of the system. However, in the debug or bringup stage of the system, this assumption may not be true. The 'MODE' bit allows the UPA to run slower than the U2S clock, which is fixed at twice the speed of SBus clock. Operations proceed normally regardless of the state of this bit and are guaranteed to work with 'MODE' set to '0'. Configuration software should set the 'MODE' bit to '1' if it determined that the UPA clock is faster than the U2S clock, which is fixed at 50 MHz in normal operation.

5.5.1.4 ECC Registers

The ECC Registers provide control, generation, and the status of ECC error related interrupts.

Table 5-34. **Physical Address of ECC Registers**

Register	Physical Address	Access Size
ECC Control Register	0x1FE 0000 0020	8 bytes
Reserved	0x1FE 0000 0028	8 bytes
Uncorrectable Error Asynchronous Fault Status Register	0x1FE 0000 0030	8 bytes
Uncorrectable Error Asynchronous Fault Address Register	0x1FE 0000 0038	8 bytes
Correctable Error Asynchronous Fault Status Register	0x1FE 0000 0040	8 bytes
Correctable Error Asynchronous Fault Address Register	0x1FE 0000 0048	8 bytes

- **ECC Control Register**
  - **Address: 0x1FE 0000 0020**
  - **Access Size: 8 bytes**

This register controls enable / disable of ECC checking and generation of ECC error related interrupts. All bits are set to '0' upon reset.

Table 5-35. **ECC Control Register Bit Definitions**

Field	Bits	Description	Type
ECC_EN	63	Enables ECC Checking. ECC Generation is always enabled.	R / W
UE_INTEN	62	Enables interrupt generation on uncorrectable error (UE).	R / W
CE_INTEN	61	Enables interrupt generation on correctable errors (CE).	R / W
Reserved	60 - 0	Reserved	

The following table shows how the ECC_EN, UE_INTEN, and CE_INTEN bits control ECC checking and error handling in U2S.

Table 5-36. **ECC Error Reporting**

ECC_EN	INTEN	Description
0	X	No ECC checking and reporting. Every UPA transaction proceeds as if there is no ECC error. Data flows through from the UPA to the SBus.
1	0	ECC checking will be done, but no interrupt will be sent on ECC error. UE on a PIO write will not be performed on the SBus. UE on a DVMA read will return an SBus error ack. An error is logged in AFSR / AFAR but no interrupt is generated. Software should clear error status before enabling interrupt.
1	1	The U2S sends an interrupt on an ECC error. UE on a PIO write will not be performed on the SBus. UE on a DVMA read will return an SBus error ack. The error is logged in the AFSR / AFAR.

- **Uncorrectable Error Asynchronous Fault Status Register (AFSR)**

- **Address: 0x1FE 0000 0030**
- **Access: 8 bytes**

Any uncorrectable error (UE) detected by the UPA interface of the U2S will log the error in the Uncorrectable Error AFSR/AFAR. Uncorrectable errors can happen during PIO write, DVMA read or DVMA partial write. Two sets of status bits are defined in this register. Bits <63:61> are the primary error status and bits <60:58> are the secondary status. One and only one of the primary error status bits can be set at any time. A primary error status bit can be set only when either

- none of the primary error conditions existed prior to this error, or
- a new error is detected at the same time that software is clearing the primary error. Error setting takes precedence over error clearing.

Secondary bits are set whenever a primary bit is set (one and only one of the primary bits can be set at a time). The secondary bits are cumulative and always indicate that information has been lost, as no address information has been captured. Setting of the primary error bits is independent.

The AFAR and bits <55:37> of the AFSR logs the address and status of the primary UE. Additional UEs will not be logged until software clears the primary error, making the AFAR and part of the AFSR available to log new errors. An interrupt is generated whenever the AFAR logs the new error address, if the interrupt is enabled.

Table 5-37. **Uncorrectable Error Asynchronous Fault Status Register (AFSR) Bit Definitions**

Field	Bits	Description	Type
P_PIO	63	Set if primary UE is caused by PIO access	R / W1C
P_DRD	62	Set if primary UE is caused by SBus DVMA read	R / W1C
P_DWR	61	Set if primary UE is caused by SBus DVMA write	R / W1C
S_PIO	60	Set if secondary UE is caused by PIO access	R / W1C
S_DRD	59	Set if secondary UE is caused by SBus DVMA read	R / W1C
S_DWR	58	Set if secondary UE is caused by SBus DVMA write	R / W1C
RESERVED	57:48	Reserved	R0
DW_OFFSET	47:45	Offset of a doubleword containing ECC error in a 64 byte block, relative to PA module 64 bytes.	R
SIZE	44:42	Size of failed primary transfer is 2^{SIZE} bytes.[a]	R
UPA_MID	41:37	UPA MID that caused the error transaction.	R
RESERVED	36:0	Reserved	R0

a. In the event that byte merging occurs at the processor (the 'E' bit is incorrectly set), such that the byte enables are not meaningful, the size returned in this field is unpredictable. The size may not reflect the actual DVMA transfer size on the SBus if the primary error is caused by a DVMA access.

- **Uncorrectable Error Asynchronous Fault Address Register (AFAR)**
  - **Address: 0x1FE 0000 0038**
  - **Access: 8 bytes**

The Uncorrectable Error Asynchronous Fault Address Register provides the physical address of the uncorrectable error transaction.

Table 5-38. **Uncorrectable Error Asynchronous Fault Address Register (AFAR) Bit Definitions**

Field	Bits	Description	Type
RESERVED	63:41	Reserved	R0
UE_PA	40:0	Physical address of error transaction	R

- **Correctable Error Asynchronous Fault Status Register (AFSR)**
  - **Address: 0x1FE 0000 0040**
  - **Access: 8 bytes**

The U2S logs correctable errors (CE) in the Correctable Error AFSR/AFAR registers. Correctable errors can happen during PIO write, DVMA read, or DVMA partial write. Two sets of status bits are defined in this register. Bits <63:61 > are the primary error status and bits <60:58> are the secondary error status. One and only one of the primary error status bits can be set at any time. A primary error status bit can be set only when:

- None of the primary error conditions existed prior to this error.
- A new error is detected at the same time software is clearing the primary error; the same time means on coincident clock cycles. Error setting takes precedence over error clearing.

Secondary bits are set whenever a primary bit is set. The secondary bits are cumulative and always indicate that information has been lost, as no address information has been captured. Setting of the primary error bits is independent.

The AFAR and bits <55:37> of the AFSR logs the address and status of the primary CE. Additional CEs will not be logged until software clears the primary error, making the AFAR and part of the AFSR available to log new errors. An interrupt is generated whenever the AFAR logs the new error address, if the interrupt is enabled.

Table 5-39. **Correctable Error Asynchronous Fault Status Register (AFSR) Bit Definitions**

Field	Bits	Description	Type
P_PIO	63	Set if primary CE is caused by PIO access.	R / W1C
P_DRD	62	Set if primary CE is caused by SBus DVMA read	R / W1C
P_DWR	61	Set if primary CE is caused by SBus DVMA write	R / W1C
S_PIO	60	Set if secondary CE is caused by PIO access	R / W1C
S_DRD	59	Set if secondary CE is caused by SBus DVMA read	R / W1C
S_DWR	58	Set if secondary CE is caused by SBus DVMA write	R / W1C
RESERVED	57:56	Reserved	R0
E_SYND	55:48	CE Syndrome bits	R
DW_OFFSET	47:45	Offset of doubleword containing ECC error in 64 byte block, relative to PA modulo 64 bytes.	R
SIZE	44:42	Size of failed primary transfer is 2^{SIZE} bytes.[a]	R
UPA_MID	41:37	UPA MID that caused the error transaction.	R
RESERVED	36:0	Reserved	R0

a. In the event that byte merging occurs (the 'E' bit is incorrectly set) at the processor, such that the byte enables are not meaningful, the size returned in this field is unpredictable. The size may not reflect the actual DVMA transfer size on the SBus if the primary error is caused by a DVMA access.

- **Correctable Error Asynchronous Fault Address Register (AFAR)**
  - **Address: 0x1FE 0000 0048**
  - **Access: 8 bytes**

The Correctable Error Asychronous Fault Address Register provides the physical address of the correctable error transaction.

Table 5-40. **Correctable Error Asynchronous Fault Address Register (AFAR) Bit Definitions**

Field	Bits	Description	Type
RESERVED	63:41	Reserved	R0
CE_PA	40:0	Physical address of error transaction	R

5.5.1.5 SBus Module Registers

These are the registers which control SBus operations. One SBus Slot Configuration Register (SSCR) exists per slot. Each SSCR contains information specific to an individual SBus slot. The SBus Control Register contains information that is common to all SBus slots supported by the U2S. All slots except slot 15 support master capability.

Table 5-41. **Physical Address of SBus Registers**

Register	Physical Address	Access Size
SBus Control Register	0x1FE 0000 2000	8 bytes
Reserved	0x1FE 0000 2008	8 bytes
SBus Asynchronous Fault Status Register	0x1FE 0000 2010	8 bytes
SBus Asynchronous Fault Address Register	0x1FE 0000 2018	8 bytes
SBus Slot 0 Configuration Register	0x1FE 0000 2020	8 bytes
SBus Slot 1 Configuration Register	0x1FE 0000 2028	8 bytes
SBus Slot 2 Configuration Register	0x1FE 0000 2030	8 bytes
SBus Slot 3 Configuration Register (MACIO #2)	0x1FE 0000 2038	8 bytes
SBus Slot 13 Configuration Register (APC)	0x1FE 0000 2040	8 bytes
SBus Slot 14 Configuration Register (MACIO #1)	0x1FE 0000 2048	8 bytes
SBus Slot 15 Configuration Register (SLAVIO)	0x1FE 0000 2050	8 bytes

- **SBus Control Register**

Further definitions of the fields contained within the SBus Control Register are provided below.

  - **Address: 0x1FE 0000 2000**
  - **Access Size: 8 bytes**

Table 5-42. **SBus Control Register Bit Definitions**

Field	Bits	Description	Type
IMPL	63:60	Implementation number of host adapter. This field is hardwired to 0x0.	R
REV	59:56	Revision number for the design; initially 0.	R
RESERVED	55:54	Reserved	R0
DMA_PERR	53:48	Set when DVMA write parity error is detected from SBus slot. Bit 53: SBus slot 14 (MACIO #1) Bit 52: SBus slot 13 (APC) Bit 51: SBus slot 3 (MACIO #2) Bit 50: SBus slot 2 Bit 49: SBus slot 1 Bit 48: SBus slot 0	R / W1C
RESERVED	47	Reserved	R0
PIO_PERR	46:40	Set when PIO load parity error occurs. These errors result in a bad ECC being delivered to the processor. The system software should check to determine whether the bad ECC comes from a data path error or a logical PIO load parity error. Bit 46: SBus slot 15 (SLAVIO) Bit 45: SBus slot 14 (MACIO #1) Bit 44: SBus slot 13 (APC) Bit 43: SBus slot 3 (MACIO #2) Bit 42: SBus slot 2 Bit 41: SBus slot 1 Bit 40: SBus slot 0	R / W1C
RESERVED	39:11	Reserved	R0
FAST_SBUS	10	Used to shorten the PIO access latency. Resets to 0.	R / W
WAKEUP_EN	9	Power Management Wakeup Enable Control. Resets to 0 0 = Disabled 1 = Enabled	R / W
ERRINT_EN	8	Enable SBus error interrupt, reset to 0. This is only for errors reported in the SBus AFSR.	R / W
RESERVED	7:6	Reserved	R0
ARB_EN	5:0	SBus DVMA arbitration enable. Reset to 0x0. Bit 5: SBus slot 14 (MACIO #1) Bit 4: SBus slot 13 (APC) Bit 3: SBus slot 3 (MACIO #2) Bit 2: SBus slot 2 Bit 1: SBus slot 1 Bit 0: SBus slot 0	R / W

IMPL

Implementation number of this host adapter for Sun4u. This is the first implementation of a Sun4u adapter.

REV

Revision number for the design.

DMA_PERR

These bits are associated with SBus master requests from both on-board and expansion locations. They are set whenever the U2S detects a DVMA write parity error from an associated master.

Note — DMA writes to the slot configuration registers which result in a parity error do not set the DMA_PERR for that slot. Instead sb_lerr_ will be generated to the master and no logging takes place. Writes with bad parity, whether to memory or UPA devices, result in bad ECC being written to the target and are logged with these bits.

Note — MACIO does not support parity.

PIO_PERR

Set when a PIO load parity error occurs. These errors result in bad a ECC being delivered to the processor. The system software should check to determine whether the bad ECC comes from a data path error or a logical PIO load parity.

FAST_SBUS

This bit controls the access latency for PIOs to the SBus. To expedite the transfer, the acknowledgment to the UPA (P_REPLY) is delivered to the UPA before the data is available. In the normal case, data will be available before the S_Reply is delivered by the System Controller to the U2S.

The FAST_SBUS bit should be set if the following condition is true (values are in nano-seconds).

– $T_{UPA} > 0.25 * T_{SBus} + 2$

T_{UPA} is the UPA clock period, and T_{SBus} is the SBus clock period.

This bit has been included for debug purposes. So long as the SBus clock is running at 25 MHz, this bit can be set to a '1' even with an 83.3 MHz (12 ns cycle time) UPA clock. Operation at 10 ns cycle time UPA clock is unknown at this time.

WAKEUP_EN

This bit is used by system software when putting various parts of the system to sleep. When set to '1', any attempt by an enabled SBus device (ones with ARB_EN == 1) to arbitrate for the SBus will result in an interrupt packet being delivered to a target for the purpose of waking up the system. Refer to the interrupt section for more details. Arbitration for SBus devices is inhibited as long as this bit is set; in other words, with WAKEUP_EN == 1 the system behaves as though ARB_EN == 0.

Additionally, this bit enables a divide by 1000 prescaler for the System Timer / Counter 0, thereby changing it from incrementing once per microsecond to once per millisecond. Timer / Counter 1 is not affected. When this bit is set the interrupt type, as indicated by the value delivered in the interrupt vector (mondo vector), is Power Management Wakeup rather than Timer / Counter Interrupt.

ERRINT_EN

When set to '1' this bit enables interrupt generation on SBus errors. Interrupts are disabled when this bit is set to '0'. Power up state is '0'.

ARB_EN

This bit enables DVMA arbitration for SBus slots 3-0, slot 13 (APC), and slot 14 (MACIO #1). If this bit is set to '0', the bus request from the slot will be ignored by the SBus arbiter.

- **SBus Asynchronous Fault Status Register (AFSR)**
  - **Address: 0x1FE 0000 2010**
  - **Access Size: 8 bytes**

The SBus AFSR/AFAR registers record PIO read / write to SBus slave device's error information. Only asynchronous errors reported through interrupt are recorded in these registers. Asynchronous errors include errors triggered by any PIO write to SBus devices, PIO read, and SBus late error.

Two sets of status bits are defined in this register. Bits <63:61> are the primary error status and bits <60:58> are the secondary error status. One and only one of the primary error status bits can be set at any time. A primary error status bit can be set only when:

- None of the primary error conditions existed prior to this error.
- A new error is detected at the same time software is clearing the primary error; the same time means on coincident clock cycles. Error setting takes precedence over error clearing.

Secondary bits are set whenever a primary bit is set. The secondary bits are cumulative and always indicate that information has been lost, as no address information has been captured. Setting of the primary error bits is independent.

The AFAR and bits <47:37> of the AFSR logs the address and status of the primary SBus PIO error. Additional SBus PIO errors will not be logged until software clears the primary error, which makes the AFAR and part of the AFSR available to log a new error. An interrupt is generated whenever the AFAR logs the new error address, if the interrupt is enabled.

Table 5-43. **SBus Asynchronous Fault Status Register (ARSR) Bit Definitions**

Field	Bits	Description	Type
P_LE	63	Set if primary error detected is Late PIO Error	R / W1C
P_TO	62	Set if primary error detected is SBus time-out (PIO wait)	R / W1C
P_BERR	61	Set if primary error detected is SBus error ack	R / W1C
S_LE	60	Set if secondary error detected is Late PIO (Error)	R / W1C
S_TO	59	Set if secondary error detected is SBus time-out (PIO wait)	R / W1C
S_BERR	58	Set if secondary error detected is SBus error ack	R / W1C
RESERVED	57:48	Reserved	R0
RD	47	0 = If primary error is caused by PIO late read 1 = If primary error is caused by PIO write	R
RESERVED	46:45	Reserved	R0
SIZE	44:42	Size of failed primary transfer is 2^{SIZE} bytes.	R
MID	41:37	UPA MID that caused error transaction	R
RESERVED	36:0	Reserved	R0

- **SBus Asynchronous Fault Address Register (AFAR)**
  - **Address: 0x1FE 0000 2018**
  - **Access: 8 bytes**

Table 5-44. **SBus Asynchronous Fault Address Register (AFAR) Bit Definitions**

Field	Bits	Description	Type
RESERVED	63:41	Reserved	R0
PA	40:0	Physical address of error transaction	R

- **SBus Slot Configuration Register**
  - **Address: 0x1FE 0000 2020 (slot 0)**
  - **Address: 0x1FE 0000 2028 (slot 1)**
  - **Address: 0x1FE 0000 2030 (slot 2)**
  - **Address: 0x1FE 0000 2038 (slot 3) - MACIO #2**
  - **Address: 0x1FE 0000 2040 (slot 13) - Audio (APC)**
  - **Address: 0x1FE 0000 2048 (slot 14) - MACIO #1**
  - **Address: 0x1FE 0000 2050 (slot 15) - SLAVIO**
  - **Access Size: 8 bytes**

All information in these registers is accessible by a PIO read / write. Part of the information is accessible by the corresponding slot DVMA master through SBus virtual address 0x4 0000 0000 when the 'BY' bit is set. The 'BY' bit can only be set by the host processor. Bits <31:00> are readable by the slot DVMA master. Only bits <26:15> are writable from the slot DVMA master. All bits in this register are reset to '0'.

Table 5-45. **SBus Slot Configuration Register Bit Definitions**

Field	Bits	Description	Type
RESERVED	63:27	Reserved	R0
SEGA	26:16	Segment Address - Provides PA <40:30> when bypass mode is used.	R / W
CP	15	Bypass mode access is cacheable	R / W
ETM	14	This slot slave interface supports Extended Transfer Mode	R / W
PE	13	Enable SBus Parity Checking - Generation is always enabled	R / W
RESERVED	12:5	Reserved	R0
BA64	4	Slave interface -- supports 64-byte burst transfer	R / W
BA32	3	Slave interface -- supports 32-byte burst transfer	R / W
BA16	2	Slave interface -- supports 16-byte burst transfer	R / W
BA8	1	Slave interface -- supports 8-byte burst transfer	R / W
BY	0	IOMMU Bypass Mode enabled	R / W

Warning — Do not set the BA8, BA16, BA32, or BA64 bits if a device is incapable of responding to that size access. For example, SLAVIO will not operate correctly if any of these bits are set.

5.5.1.6 IOMMU Registers

Table 5-46. **Physical Address of IOMMU Registers**

Register	Physical Address	Access Size
IOMMU Control Register	0x1FE 0000 2400	8 bytes
TSB Base Address Register	0x1FE 0000 2408	8 bytes
IOMMU Flush Register	0x1FE 0000 2410	8 bytes
SBus Virtual Address Diagnostics Register	0x1FE 0000 4400	8 bytes
TLB Tag Compare Diagnostics Register	0x1FE 0000 4408	8 bytes
IOMMU LRU Queue Diagnostics Register	0x1FE 0000 4500 - 0x1FE 0000 457F	8 bytes
TLB Tag Diagnostics Register	0x1FE 0000 4580 - 0x1FE 0000 45FF	8 bytes
TLB Data RAM Diagnostics Register	0x1FE 0000 4600 - 0x1FE 0000 46FF	8 bytes

- **IOMMU Control Register**
  - **Address: 0x1FE 0000 2400**
  - **Access Size: 8 bytes**

The IOMMU Control Register provides means to enable and disable the diagnostic mode, TSB size, and page size. It also contains revision control information.

Note — This register physically resides in the SBus module, so it will not provide synchronization with any other IOMMU register. Similarly, on writes to this register, software will need to either read this register or another in the SBus module to guarantee write completion.

Table 5-47. **IOMMU Control Register Bit Definitions**

Field	Bits	Description	Type
IMPL	63:60	IOMMU implementation number, hardwired to 0	R
REV	59:56	IOMMU revision number of current implementation; 0 for first revision	R
RESERVED	55:19	Reserved	R0
TSB_SIZE	18:16	TSB table size measured in the number of entries. Each entry is 8 bytes. 000 = 1 Kbyte 001 = 2 Kbyte 010 = 4 Kbyte 011 = 8 Kbyte 100 = 16 Kbyte 101 = 32 Kbyte 110 = 64 Kbyte 111 = 128 Kbyte	R / W

Table 5-47. **IOMMU Control Register Bit Definitions** (Continued)

Field	Bits	Description	Type
RESERVED	15:3	Reserved	R0
TBW_SIZE[a]	2	Assumed page size during TSB lookup. 0 = 8 Kbyte page 1 = 64 Kbyte page	R / W
MMU_DE	1	Diagnostic mode enable, when set it enables the diagnostic mode. See *Table 5-57 TLB Tag Diagnostics Register Bit Definitions* on page 5-38.	R / W
MMU_EN	0	IOMMU enable bit, when set it enables the translation.	R / W

a. If DVMA mappings are always 8 Kbyte pages, or mixed 8 Kbyte and 64 Kbyte pages, set this bit to '0' so that the index is constructed for 8 Kbyte lookup. If all DVMA mappings are to 64 Kbyte pages, set this bit to '1' so that the index is based on 64 Kbyte pages. When this bit is '0', a 64 Kbyte mapping should be placed in all 8 TSB entries in which it is indexed.

Address space size and base address are controlled by TSB_SIZE and TBW_SIZE as shown in *Table 5-48.*

Table 5-48. **Address Space Size and Base Address Determination**

TSB_SIZE	TBW_SIZE = 0			TBW_SIZE = 1		
	VA Space Size	VABase Address[a]	TSB Index[3]	VA Space Size	VA Base Address[a]	TSB Index[3]
0	8 Mbyte	0xFF80 0000	VA <22:13>	64 Mbyte	0xFC00 0000	VA <25:16>
1	16 Mbyte	0xFF00 0000	VA <23:13>	128 Mbyte	0xF800 0000	VA <26:16>
2	32 Mbyte	0xFE00 0000	VA <24:13>	256 Mbyte	0xF000 0000	VA <27:16>
3	64 Mbyte	0xFC00 0000	VA <22:13>	512 Mbyte	0xE000 0000	VA <28:16>
4	128 Mbyte	0xF800 0000	VA <25:13>	1Gbyte	0xC000 0000	VA <29:16>
5	256 Mbyte	0xF000 0000	VA <27:13>	2 Gbyte	0x8000 0000	VA <30:16>
6	512 Mbyte	0xE000 0000	VA <28:13>	4 Gbyte	0x0000 0000	VA <31:16>
7	1Gbyte	0xC000 0000	VA <29:13>	N/A	N/A	N/A

a. Software should use the value shown. It has been chosen to clearly eliminate aliasing. No hardware checks are performed to prevent aliasing.

For reference, the IOMMU Translation Table Entry (TTE) format has been included in *Table 5-49.*

- **IOMMU Translation Table Entry (TTE)**

Table 5-49. **IOMMU Translation Table Entry Bit Definitions**

Field	Bits	Description	Type
DATA_V	63	Valid; Indicates that the TTE is valid	R / W
RESERVED	62	Reserved	R0
DATA_SIZE	61	Page Size 0 = 8 Kbyte 1 = 64 Kbyte	R / W

Table 5-49. **IOMMU Translation Table Entry Bit Definitions** (Continued)

Field	Bits	Description	Type
STREAM	60	If set, the page is streamable.	R / W
LOCAL_BUS	59	If set, the access is to the same bus segment.	R / W
DATA_SOFT_2	58:51	Assigned for software use.	R / W
RESERVED	50:41	Reserved; Room for growth, variable physical address size.	R0
DATA_PA<N1:13>	40:13	Physical Page Number	R / W
DATA_SOFT	12:7	Assigned for software use.	R / W
RESERVED	6:5	Reserved	R0
CACHEABLE	4	0 = Access to Noncacheable space 1 = Access to Cacheable space	R / W
RESERVED	3:2	Reserved	R0
DATA_W	1	Writable page; Attempts to write to a page with this bit set to '0' will result in an error ack.	R / W
RESERVED	0	Reserved	R0

In *Table 5-50*, while software should replace 'X' with '0' for a legal code, hardware will interpret combinations of the IOMMU TTE STREAM, LOCAL_BUS, and CACHEABLE bits as follows:

Table 5-50. **IOMMU TTE STREAM, LOCAL_BUS, and CACHEABLE Bit Combinations**

STREAM <60>	LOCAL_BUS <59>	CACHEABLE <4>	Meaning
X	1	X	Access to slave on same I/O bus segment
0	0	0	Non-cacheable access system
0	0	1	Consistent-mode cacheable access to system memory
1	0	X	Streaming-mode cacheable access to system memory

The physical address of a DVMA transaction is generated based on the value of the 'MMU_EN' bit, the 'BY' bit of the SBus Slot Configuration Register, and the SBus Virtual Address bits <31:30>. *Table 5-51* shows the various IOMMU modes.

Table 5-51. **IOMMU Modes of Operation**

MMU_EN	BY	VA <31:30>	Mode
X	1	00	Bypass
X	1	01	Slot Configuration Register access
0	0	XX	Pass-through
0	1	1X	Pass-through
1	0	XX	Translation
1	1	1X	Translation

In Bypass mode the physical address is formed by concatenating SEGA<40:30> with SBus Virtual Address <29:0>. In pass-through mode the physical address is formed by appending 9 or 10 bits of zero onto bits <31:0> or <30:0> of the SBus Virtual Address. In pass-through mode accesses are only to memory space and are performed only in consistent mode. The access is considered cacheable and a coherent operation will be generated to the UPA. In translation mode the physical address is obtained by performing virtual address to physical address translation through the TLB.

The SBus Slot Configuration Register can be accessed directly by an SBus master through VA<31:30> == 01 while the bypass enable bit 'BY' is set to '1'.

- **TSB Base Address Register**
  - **Address: 0x1FE 0000 2408**
  - **Access Size: 8 bytes**

The TSB Base Address Register contains the pointer to the first-entry of the TSB table. Together with part of the virtual address it uniquely identifies the address where hardware should fetch the TTE from TSB table. The TSB table has to be aligned on an 8 Kbyte boundary. The lower order 13 bits are assumed to be 0x0 during TSB table lookup. Tables larger than 8 Kbytes are only constrained to be on 8 Kbyte boundaries rather than being size aligned.

Table 5-52. **TSB Base Address Register Bit Definitions**

Field	Bits	Description	Type
RESERVED	63:41	Reserved	R0
TSB_BASE	40:13	Upper 28 bits of the SBus TSB's physical address	R / W
RESERVED	12:0	Reserved	R0

- **Flush Address Register**
  - **Address: 0x1FE 0000 2410**
  - **Access Size: 8 bytes**

The Flush Address Register is a write-only pseudo-register that allows software to perform address based flushes of a mapping from the TLB. The data written to this address contains the page number to be flushed. A TLB entry with a matched page number will be invalidated.

Table 5-53. **IOMMU Flush Register Bit Definitions**

Field	Bits	Description	Type
RESERVED	63:32	Reserved; write has no effect	W
FLUSH_VPN	31:13	31:16 = virtual page number if 64 Kbyte page 15:13 = don't care (if 64 Kbyte page) 31:13 = virtual page number if 8 Kbyte page	W
RESERVED	12:0	Reserved; write has no effect	W

Note — On any cycle, an entry may be looked up in the TLB either by the flush hardware, or the DVMA hardware. Simultaneous flush and use cannot occur. Devices attempting to use an entry which software has flushed will either get the entry before the flush if the request beats the flush, or will find the entry missing from the TLB and will perform a hardware tablewalk. Software should invalidate the entry from the TSB before issuing the flush.

Note — Completion of any read to the IOMMU (except the IOMMU Control Register) within the U2S guarantees that the flush transaction has completed.

- **SBus Virtual Address Diagnostics Register**
  - **Address: 0x1FE 0000 4400**
  - **Access Size: 8 bytes**

The SBus Virtual Address Diagnostics Register is used to set up the virtual address for TLB compare diagnostics. The virtual address is written to this register and the compare results from the TLB can be read.

Table 5-54. **SBus Virtual Address Diagnostics Register Bit Definitions**

Field	Bits	Description	Type
RESERVED	63:32	Reserved	R0
SBUS_VPN	31:13	SBus virtual page number	R / W
RESERVED	12:0	Reserved	R0

- **TLB Tag Compare Diagnostics Register**
  - **Address: 0x1FE 0000 4408**
  - **Access Size: 8 bytes**

The TLB Tag Compare Diagnostics Register compares the given instruction Tag with the on-chip TLB. It registers a 'hit' if the instruction Tag is present and a 'miss' if it is absent.

Table 5-55. **TLB Tag Compare Diagnostics Register Bit Definitions**

Field	Bits	Description	Type
RESERVED	63:16	Reserved	R0
SBUS_VPN	15:0	TLB tag comparator output for each entry.	R

Note — The TLB Tag Compare Diagnostics Register provides a diagnostics path to the 16-entry TLB Tag when the 'MMU_DE' bit <1> in the IOMMU Control Register is set. Bit 0 represents the comparison result of the first TLB Tag entry, and bit 15 represents the last.

In order to avoid invalid address translation after TLB diagnostics, the valid bits in the TLB should be reset appropriately before doing any meaningful address translation. A diagnostics write to a read-only space or a read from a write-only space will be ignored.

- **IOMMU LRU Queue Diagnostics Register**
  - **Address: 0x1FE 0000 4500 - 0x1FE 0000 457F**

– **Access Size: 8 bytes**

The IOMMU LRU Queue Diagnostics Register can be directly accessed by PIO read for diagnostic purposes. The 'MMU_DE' bit in IOMMU Control Register must be set to perform direct access. There are 16 entries in the LRU Queue. Each entry contains a unique value range from 0x0 to 0xF. Entry 0 contains a pointer to the TLB entry least recently used. Entry 15 contains a pointer to the TLB entry most recently used.

Table 5-56. **IOMMU LRU Queue Diagnostics Register Bit Definitions**

Field	Bits	Description	Type
RESERVED	63:4	Reserved	R0
LRU_DO	3:0	LRU entry selected.	R

- **TLB Tag Diagnostics Register**
  - **Address: 0x1FE 0000 4580 - 0x1FE 0000 45FF**
  - **Access Size: 8 bytes**

The TLB Tag Diagnostics Register provides a diagnostics path to the 16-entry TLB Tag when the 'MMU_DE' bit <1> in the IOMMU Control Register is turned on.

Table 5-57. **TLB Tag Diagnostics Register Bit Definitions**

Field	Bits	Description	Type
RESERVED	63:22	Reserved	R0
W	21	Writable bit. When set, the p age mapped by the TLB has write permission granted.	R / W
S	20	Stream Bit 0 = page is not streamable 1 = page is streamable	R / W
SIZE	19	Page Size 0 = 8 Kbyte 1 = 64 Kbyte	R / W
VPN	18:0	VPN[31:13]	R / W

Note — Diagnostic accesses should insure that multiple match conditions are not generated. The result of multiple matches is unpredictable.

- **TLB Data RAM Diagnostics Register**
  - **Address: 0x1FE 0000 4600 - 0x1FE 0000 46FF**
  - **Access: 8 bytes**

The TLB Data RAM Diagnostics Register provides direct PIO accesses to the 16 entries of TLB Data RAM. The 'MMU_DE' bit <1> in the IOMMU Control Register must be set to perform the accesses.

Table 5-58. **TLB Data RAM Diagnostics Register Bit Definitions**

Field	Bits	Description	Type
RESERVED	63:31	Reserved	R0
V	30	Valid bit, when set, the TLB data field is meaningful	R / W
L	29	Local_Bus to indicate the direction of DVMA transfer. 0 = non-local access (DVMA to memory or the UPA) 1 = local access to U2S (intra SBus to SBus transfer)	R / W
C	28	Cacheable; access type is strictly determined by this bit, not by PA<40> 0 = non-cacheable 1 = cacheable access	R / W
PA[40:13]	27:0	VPN[31:13]	R / W

5.5.1.7 Streaming Buffer Registers

This section describes the registers associated with the Streaming Buffer. This block inside the U2S is used for speeding up DMA transfers by providing read ahead and write behind buffering of data. Streaming DMA enters the coherence domain at a different time than consistent transfers with reads leaving the coherent space earlier and writes entering the coherent space later.

Note — While the Streaming Buffer is a separate block inside U2S, its Streaming Buffer Control Register logically resides in the SBus module. This means that an access designed to assure write completion should either access a streaming buffer register or one in the SBus module.

Table 5-59. **Physical Address of Streaming Buffer Registers**

Register	Physical Address	Access Size
Streaming Buffer Control Register	0x1FE 0000 2800	8 bytes
Streaming Buffer Page Flush / Invalidate Register	0x1FE 0000 2808	8 bytes
Streaming Buffer Flush Synchronization Register	0x1FE 0000 2810	8 bytes
Streaming Buffer Data RAM Diagnostic Register	0x1FE 0000 5000 - 0x1FE 0000 53FF	8 bytes
Streaming Buffer Error Status Diagnostics Register	0x1FE 0000 5400 - 0x1FE 0000 57FF	8 bytes
Streaming Buffer Page Tag Diagnostics Register	0x1FE 0000 5800 - 0x1FE 0000 587F	8 bytes
Streaming Buffer Line Tag Diagnostics Register	0x1FE 0000 5900 - 0x1FE 0000 597F	8 bytes

The streaming buffer performs all operations on 8 Kbyte pages. Transfers larger than 8 Kbyte, even though they may be mapped with a single IOMMU entry, require 8 flush operations.

- **Streaming Buffer Control Register**
  - **Address: 0x1FE 0000 2800**
  - **Access Size: 8 bytes**

The Streaming Buffer Control Register contains the implementation and revision number of the host adapter. It also enables / disables the streaming buffer and the diagnostic mode.

Table 5-60. **Streaming Buffer Control Register Bit Definitions**

Field	Bits	Description	Type
IMPL	63:60	Implementation number of this host adapter	R
REV	59:56	Revision number for this design	R
RESERVED	55:2	Reserved	R0
DE	1	Diagnostic Mode enable. Set to '1' to enable diagnostic mode access. This bit is reset to '0'.	R / W
SB_EN	0	Streaming buffer enable / disable. Set to '1' to enable Streaming buffer. This bit is reset to '0'.	R / W

- **Streaming Buffer Page Flush / Invalidate Register**
  - **Address: 0x1FE 0000 2808**
  - **Access Size: 8 bytes**

The Streaming Buffer Page Flush / Invalidate Register is a write-only pseudo register. It provides a means for software to invalidate / flush an entry in the streaming buffer with a matching tag. The data written to this address contains the virtual page number to be used for match comparison. The flush / invalidation is based on an 8 Kbyte page size.

Table 5-61. **Streaming Buffer Page Flush / Invalidate Register Bit Definitions**

Field	Bits	Description	Type
FLUSH_A	31:13	8 Kbyte virtual page to be flushed / invalidated	W
RESERVED	12:0	Reserved	R0

- **Streaming Buffer Flush Synchronization Register**
  - **Address: 0x1FE 0000 2810**
  - **Access Size: 8 bytes**

The Streaming Buffer Flush Synchronization Register provides a means for software to determine when flush data has entered the coherent memory domain. Data written to this register contains the physical address of the flush flag. Writing to this register triggers the U2S to set the flush flag, a 32-bit value, to 0x1 when it has completed all in progress flush operations. The low order 2 bits of the FLAG_PA address will be ignored. Please read Chapter 8, Streaming Buffer, for more information of how the synchronization is completed.

Note — Properly functioning hardware MUST set the flush flag within 0.5 seconds. Software may use this value as a time-out to protect against catastrophic failure.

Table 5-62. **Streaming Buffer Flush Synchronization Register Bit Definitions**

Field	Bits	Description	Type
FLAG_PA	40:2	Word aligned physical address for synchronization update	W
RESERVED	1:0	Reserved	R0

The FLAG_PA must point to a valid cacheable address. No hardware checking is done to insure this and the hardware will generate a cacheable transaction regardless of the address. If the address is non-cacheable, the error would be logged in the System Controller using the Sysio_Status Register 'IADDR' bit.

- **Streaming Buffer Data RAM Diagnostics Register**
  - **Address: 0x1FE 0000 5000 - 0x1FE 0000 53FF**
  - **Access Size: 8 bytes**

There are sixteen 64-byte entries in the Streaming Buffer. For data and error status accesses, bits <9:6> of the address select the buffer number and bits <5:3> selects the 8-byte entry (or 1 bit error status) within the buffer.

Table 5-63. **Streaming Buffer Data RAM Diagnostics Register Bit Definitions**

Field	Bits	Description	Type
DRDA	63:0	Data	R / W

- **Streaming Buffer Error Status Diagnostics Register**
  - **Address: 0x1FE 0000 5400 - 0x1FE 0000 57FF**
  - **Access Size: 8 bytes**

Each entry of the Streaming Buffer Error Status Diagnostics Register has 8 error bits associated with it. Each error bit represents the error status of 8 bytes of data. These bits are only visible to software under diagnostic mode. Each error bit is accessed individually.

Table 5-64. **Streaming Buffer Error Status Diagnostics Register Bit Definitions**

Field	Bits	Description	Type
DRER	0	UPA read reply error bit	R / W

- **Streaming Buffer Page Tag Diagnostics Register**
  - **Address: 0x1FE 0000 5800 - 0x1FE 0000 587F**
  - **Access Size: 8 bytes**

The Page Tags are directly accessible through PIO access. This can be done only when the 'DE' bit <1> of the Streaming Buffer Control Register is set to '1'.

Table 5-65. **Streaming Buffer Page Tag Diagnostics Register Bit Definitions**

Field	Bits	Description	Type
PTPA	48:21	Physical page number (as an 8 Kbyte page)	R / W
PTVA	20:2	Virtual page number (as an 8 Kbyte page)	R / W
PTVD	1	Valid bit for page	R / W
PTRD	0	Read / Write bit for page	R / W

Caution — Valid bits on all entries should be reset to '0' after finishing diagnostics of the Streaming Buffer Page Tag.

- **Streaming Buffer Line Tag Diagnostics Register**
  - **Address: 0x1FE 0000 5900 - 0x1FE 0000 597F**
  - **Access Size: 8 bytes**

The Streaming Buffer Line Tag Diagnostics Register contains information related to the line in the Streaming Buffer. This information can be directly accessed when the Streaming Buffer is in diagnostic mode, and the 'DE' bit <1> is set to '1' in the Streaming Buffer Control Register.

Table 5-66. **Streaming Buffer Line Tag Diagnostics Register Bit Definitions**

Field	Bits	Description	Type
LTSP	20:15	Start pointer for dirty data portion of buffer.	R / W
LTLA	14:8	The line offset address for this entry is within an 8 Kbyte page. It corresponds to VA <12:6> from the SBus access. The VPN is stored in the page tag array.	R / W
LTEP	7:2	End pointer (+1) for dirty data portion	R / W
LTVD	1	Valid bit for line	R / W
LTFH	0	Fetch Outstanding / Flush Necessary bit	R / W

The LTEP field should be set to '0' if the page is readable. If the page is writable, this field should be set to one greater than the end byte address of the dirty data chunk in the data ram (modulo buffer size of 64 Kbytes).

Caution — Valid bits on all entries should be reset to '0' after finishing diagnostics of the Streaming Buffer Line Tag.

5.5.1.8 Interrupt State Registers

All Interrupt State Registers are accessed as 64-bit words. Writing to an unused data field has no effect. Unspecified data bits are read back with '0'. Three tables, the Physical Address of Interrupt Mapping Registers (see *Table 5-67* on page 5-42), the Physical Address of Clear Interrupt Registers (see *Table 5-69* on page 5-46), and the Physical Address of Interrupt State Diagnostic Registers (see *Table 5-72* on page 5-47) list the addresses of the Interrupt State Registers.

Note — MACIO #2 Ethernet interrupt is available at SBus slot 3, interrupt 4. MACIO #2 SCSI interrupt is available at SBus Slot 3, interrupt 3.

- **Interrupt Mapping Registers**

Table 5-67. **Physical Address of Interrupt Mapping Registers**

Register	Physical Address	Access Size
SBus Slot 0 Interrupt Mapping Register	0x1FE 0000 2C00	8 bytes
SBus Slot 1 Interrupt Mapping Register	0x1FE 0000 2C08	8 bytes

Table 5-67. **Physical Address of Interrupt Mapping Registers** (Continued)

Register	Physical Address	Access Size
SBus Slot 2 Interrupt Mapping Register	0x1FE 0000 2C10	8 bytes
SBus Slot 3 Interrupt Mapping Register	0x1FE 0000 2C18	8 bytes
SCSI Interrupt Mapping Register	0x1FE 0000 3000	8 bytes
Ethernet Interrupt Mapping Register	0x1FE 0000 3008	8 bytes
Parallel Port Interrupt Mapping Register	0x1FE 0000 3010	8 bytes
Audio Interrupt Mapping Register	0x1FE 0000 3018	8 bytes
Power Fail Interrupt Mapping Register	0x1FE 0000 3020	8 bytes
Keyboard / Mouse / Serial Interrupt Mapping Reg.	0x1FE 0000 3028	8 bytes
Floppy Interrupt Mapping Register	0x1FE 0000 3030	8 bytes
Thermal Warning Interrupt Mapping Register	0x1FE 0000 3038	8 bytes
Keyboard Interrupt Mapping Register[a]	0x1FE 0000 3040	8 bytes
Mouse Interrupt Mapping Register[a]	0x1FE 0000 3048	8 bytes
Serial Interrupt Mapping Register[a]	0x1FE 0000 3050	8 bytes
Timer 0 Interrupt Mapping Register	0x1FE 0000 3060	8 bytes
Timer 1 Interrupt Mapping Register	0x1FE 0000 3068	8 bytes
Uncorrectable Error Interrupt Mapping Register	0x1FE 0000 3070	8 bytes
Correctable Error Interrupt Mapping Register	0x1FE 0000 3078	8 bytes
SBus Asynchronous Error Interrupt Mapping Reg.	0x1FE 0000 3080	8 bytes
Power Mgmt Wakeup Interrupt Mapping Register	0x1FE 0000 3088	8 bytes
UPA Expansion (Graphics) Interrupt Mapping Reg.	0x1FE 0000 3090 and 0x1FE 0000 6000[b]	8 bytes
Reserved Interrupt Mapping Register	0x1FE 0000 3098 and 0x1FE 0000 8000[c]	8 bytes

a. The keyboard, mouse, and serial interrupts are defined for future devices which do not combine all of the interrupts into one.

b. Accesses to either of these addresses behave identically; in other words, the registers are double mapped.

c. This interrupt is reserved for a UPA slave interrupt device and should not be used for a general purpose interrupt.

Interrupts delivered to the processor by the U2S have the format shown below in *Figure 5-3*.

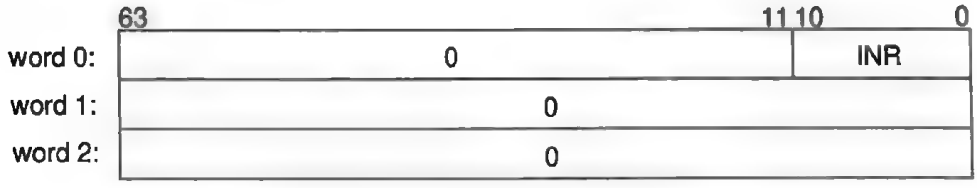


Figure 5-3. **U2S Interrupt Format**

The INR is an 11-bit interrupt number which indicates the cause of the interrupt. Where possible, the interrupt is precise (i.e., it points to only one interrupt source). This permits the dispatch of the proper interrupt service routine without any register polling. Bits <63:11> of Word 0 are guaranteed to be '0' for all U2S generated interrupts. Software can use this knowledge to distinguish these interrupts from others such as cross-calls. Words 1 and 2 are guaranteed to be '0'.

To conform to the format shown in *Figure 5-3*, interrupt mapping registers are supplied to specify those bits of the INR which are not hard coded. The two formats of mapping registers are shown below in *Figure 5-4*. For the partial format, the lower 6 bits are not writable and return the appropriate INO when read. In the case of the SBus interrupts, the INO returned is '0'. The value of the IGN comes from bits <50:46> of the U2S Control Register (see *§ 5.5.1.3 U2S Control / Status Register* on page 5-24 for details).

Note — All U2S internal interrupts and SBus interrupts use the partial format. The full format is only used by the graphics interrupts.

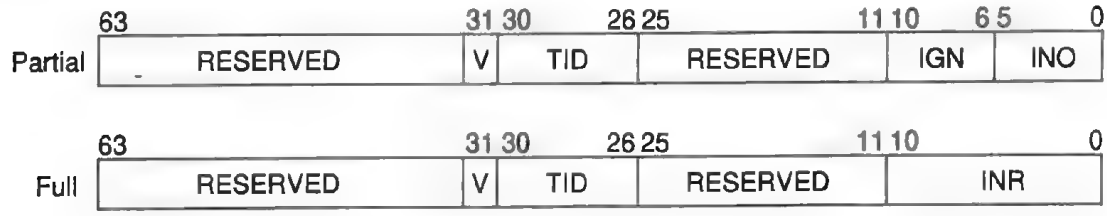


Figure 5-4. **Partial and Full Interrupt Mapping Formats**

Graphics interrupts use the full format because the U2S is acting only as the interrupt concentrator for the graphics interrupt. Logically, the interrupt and its associated state reside in the graphics device. Only the mapping register exists in the U2S. Having full access to the INR enables software to provide consistency in programming the upper bits for various devices. For example, software might put the PortID of a device in these upper bits. Since the graphics device is a different port than the U2S, the value can be programmed independently.

V (Read / Write)

Valid bit. When this bit is set to '0', it inhibits the dispatch of the interrupt to the processor an does not affect the machine state. In other words, an interrupt in the transmit state will stay there as long as the 'Valid' bit is set to '0' without sending the interrupt. As soon as the 'Valid' bit is set to '1', the interrupt will be delivered. Interrupts in the pending state can still be forced back to idle by writing the interrupt pending register regardless of the state of the 'Valid' bit. The power up state of this bit is '0'.

TID (Read / Write)

Target ID. This is the UPA PortID (PID) of the target which is to receive the interrupt.

IGN (Read)

Interrupt Group Number. This bit is not writable through the mapping registers. It is set in the U2S Control Register.

INO (Read)

Interrupt Number Offset. *Table 5-68* lists the INO for the on-board I/O devices and error interrupts.

Table 5-68. **Interrupt Number Offset Assignments**

Interrupt Number Offset (INO)	Interrupt Source
000nnn	SBus slot 0 (nnn = level 1 through 7)
001nnn	SBus slot 1 (nnn = level 1 through 7)
010nnn	SBus slot 2 (nnn = level 1 through 7)
011nnn	SBus slot 3 (nnn = level 1 through 7)
100000	SCSI
100001	Ethernet
100010	Parallel Port
100100	Audio (APC)
100101	Power Fail
101000	Keyboard / Mouse / Serial Ports
101001	Floppy
101010	Thermal Warning
101011	Keyboard[a]
101100	Mouse[a]
101101	Serial[a]
110000	Timer / Counter 0
110001	Timer / Counter 1
110100	Uncorrectable Errors
110101	Correctable Errors
110110	SBus Asynchronous Error
110111	Power Management Wakeup
111111	Reserved

a. These interrupt numbers are reserved for a future implementation in which the keyboard, mouse, and serial interrupts are not wire-ORed together. They are not active in this implementation.

Note — For the SBus interrupts, there is only one mapping register per bus. However, because the interrupt level is part of the value delivered in the interrupt vector, SBus interrupts are still precise.

INR (Read / Write)

Interrupt Number. The full field is programmable for the on board graphics and UPA slave slot interrupts.

- **Clear Interrupt Registers**

The Clear Interrupt Pseudo-Registers are listed in *Table 5-69* below.

Table 5-69. **Physical Address of Clear Interrupt Pseudo-Registers**

Register	Physical Address	Access Size
SBus Slot 0 Clear Interrupt Register	0x1FE 0000 3408 - 0x1FE 0000 3438	8 bytes
SBus Slot 1 Clear Interrupt Register	0x1FE 0000 3448 - 0x1FE 0000 3478	8 bytes
SBus Slot 2 Clear Interrupt Register	0x1FE 0000 3488 - 0x1FE 0000 34B8	8 bytes
SBus Slot 3 Clear Interrupt Register	0x1FE 0000 34C8 - 0x1FE 0000 34F8	8 bytes
SCSI Clear Interrupt Register	0x1FE 0000 3800	8 bytes
Ethernet Clear Interrupt Register	0x1FE 0000 3808	8 bytes
Parallel Port Clear Interrupt Register	0x1FE 0000 3810	8 bytes
Audio Clear Interrupt Register	0x1FE 0000 3818	8 bytes
Power Fail Clear Interrupt Register	0x1FE 0000 3820	8 bytes
Keyboard / Mouse / Serial Clear Interrupt Register	0x1FE 0000 3828	8 bytes
Floppy Clear Interrupt Register	0x1FE 0000 3830	8 bytes
Thermal Warning Clear Interrupt Register	0x1FE 0000 3838	8 bytes
Keyboard Clear Interrupt Register[a]	0x1FE 0000 3840	8 bytes
Mouse Clear Interrupt Register[a]	0x1FE 0000 3848	8 bytes
Serial Clear Interrupt Register[a]	0x1FE 0000 3850	8 bytes
Timer 0 Clear Interrupt Register	0x1FE 0000 3860	8 bytes
Timer 1 Clear Interrupt Register	0x1FE 0000 3868	8 bytes
Uncorrectable Error Clear Interrupt Register	0x1FE 0000 3870	8 bytes
Correctable Error Clear Interrupt Register	0x1FE 0000 3878	8 bytes
SBus Asynchronous Error Clear Interrupt Register	0x1FE 0000 3880	8 bytes
Power Mgmt Wakeup Clear Interrupt Register	0x1FE 0000 3888	8 bytes

a. The keyboard, mouse, and serial interrupts are defined for future devices which do not combine all of the interrupts into one. They are not active in this implementation.

A register exists for each interrupt source. The lower 2 bits of the data word written to this register specify the type of operation as shown in *Table 5-70* below. All other bits should be written as '0' to guarantee future compatibility. Reads of these registers return '0'.

Table 5-70. **Clear Interrupt Pseudo-Register**

Data Value	Description
0x0	Transition the state of the machine from any state to idle.
0x1	Transition the state of the machine from any state to transmit.
0x2	Reserved
0x3	Transition the state of the machine from any state to pending.

Note — Interrupts can be forced by writing 0x1 to the appropriate clear interrupt pseudo register. To determine the interrupt state, use the interrupt state diagnostic registers.

- **Interrupt Re-entry Timer Register**
  - **Address: 0x1FE 0000 2C20**
  - **Access Size: 8 bytes**

After an interrupt packet receives a NACK from the System Controller, the U2S will wait for a certain number of clocks and reissue the packet again. This register controls the number of clocks the interrupt dispatch unit should wait before re-issuing the interrupt packet. The count specified by this register is not precise: it is a free running counter which the logic samples. It must roll through '0' twice before the packet is retried.

Table 5-71. **Interrupt Retry Timer Register Bit Definitions**

Field	Bits	Description	Type
LIMIT	7:0	Limit; the retry interval	R / W

Note — The Interrupt Retry Timer Register provides a maximum of (255 or 256) clocks of delay before reissuing the interrupt to the UPA. The maximum delay is approximately 10 μsec using the internal U2S clock for a reference source (5.1 μsec per iteration through the counter with a worst case of nearly two complete cycles counting to 0). The minimum delay is (LIMIT + 1) clock cycles. The default SBus clock used for this timing calculation is 25 MHz. The value would need to be scaled if a different clock frequency is used.

- **Interrupt State Diagnostic Registers**

Table 5-72. **Physical Address of Interrupt State Diagnostic Registers**

Register	Physical Address	Access Size
SBus Interrupt State Diagnostics Register	0x1FE 0000 4800	8 bytes
OBIO and Misc Interrupt State Diagnostics Reg.	0x1FE 0000 4808	8 bytes

Each interrupt input has an Interrupt State Register associated with it. This state register can be either of type 'level' or of type 'pulse'.

In the 'level' sensitive case, three states are present: idle, transmit, and pending. Idle represents the state where no interrupts are reported. Transmit indicates that an interrupt has been detected and should be delivered to the processor if the 'Valid' bit is set for the mapping register.

'Pending' is the state when the interrupt has been delivered to the processor and subsequent interrupt conditions are filtered (ignored) until the software pushes the machine state back to idle.

In the 'pulse' case, only two states are present: idle and transmit. Idle represents the state where no interrupts are reported. Transmit indicates that an interrupt has been detected and should be delivered to the processor if the 'Valid' bit is set for the mapping register. No pending state is present so the machine state transitions from transmit back to idle.

The interrupt dispatch unit uses information in these registers to control the dispatching of interrupts. Diagnostic access is provided to allow software to read the state of each interrupt source. The clear interrupt registers provide individual write paths to the interrupt sources. Please read Chapter 6, Interrupts, for a more detailed description about interrupt dispatching and state transition. The meaning of the state bits and their layout are shown in *Table 5-73*, Interrupt State Meanings, below. For the case of the SBus Interrupt Diagnostics Registers the formula for computing the location of the interrupt state is given by

– Int_State (slot m, level n) is at location (16m + 2n + 1) : (16m + 2n)

Table 5-73. **Interrupt State Meanings**

Field	Description
INT_STATE <1:0>	Interrupt State 00 = Idle state; no interrupt received or pending 01 = Transmit state; interrupt is received but not dispatched 10 = Illegal state 11 = Pending state; interrupt is received and dispatched

Bit definitions of the SBus Interrupt Diagnostics Register are shown in a general way in *Table 5-74* below. Refer to the formula above for specific bit positions. As an example, the bit position for SBus slot 1, level 2 is <21:20>.

SBus Interrupt State Diagnostic Register

– **Address: 0x1FE 0000 4800**

– **Access Size: 8 bytes**

The SBus Interrupt State Diagnostic Register provides information concerning the interrupt state of SBus slots 3 - 0, levels 1-7.

Table 5-74. **SBus Interrupt Diagnostics Register Bit Definitions**

Bits	Description	Type
63:50	SBus slot 3 levels 1-7	R
49:48	Reserved	R0
47:34	SBus slot 2 levels 1-7	R
33:32	Reserved	R0
31:18	SBus slot 1 levels 1-7	R
17:16	Reserved	R0
15:2	SBus slot 0 levels 1-7	R
1:0	Reserved	R0

On-Board I/O and Miscellaneous Interrupt State Diagnostic Register

– **Address: 0x1FE 0000 4808**

– **Access Size: 8 bytes**

Table 5-75. **On Board I/O and Miscellaneous Interrupt Diagnostics Register Bit Definitions**

Bits	Description	Type
63:36	Reserved	R0
35	Expansion UPA Interrupt State	R
34	Reserved Interrupt State	R
33:32	Power Management Wakeup Interrupt State	R
31:30	SBus Error Interrupt State	R
29:28	Correctable Error Interrupt State	R
27:26	Uncorrectable Error Interrupt State	R
25:24	Timer / Counter 1 Interrupt State	R
23:22	Timer / Counter 0 Interrupt State	R
21:20	Serial Interrupt State	R
19:18	Mouse Interrupt State	R
17:16	Keyboard Interrupt State	R
15:14	Thermal Warning Interrupt State	R
13:12	Floppy Interrupt State	R
11:10	Keyboard / Mouse / Serial Interrupt State	R
9:8	Power Fail Interrupt State	R
7:6	Audio Interrupt State	R
5:4	Parallel Port Interrupt State	R
3:2	Ethernet Interrupt State	R
1:0	SCSI Interrupt State	R

5.5.1.9 Timer / Counter Registers

Table 5-76. **Physical Address of Timer / Counter Registers**

Register	Physical Address	Access Size
Timer / Counter 0 Count Register	0x1FE 0000 3C00	8 bytes
Timer / Counter 0 Limit Register	0x1FE 0000 3C08	8 bytes
Timer / Counter 1 Count Register	0x1FE 0000 3C10	8 bytes
Timer / Counter 1 Limit Register	0x1FE 0000 3C18	8 bytes

Note — The WAKEUP_EN bit <9> in the SBus Control Register results in an increment once per 1000 clocks rather than once per clock for Timer / Counter 0. Timer / Counter 1 is unaffected by WAKEUP_EN.

- **Timer / Counter Count Registers**
  - **Address: 0x1FE 0000 3C00 and 0x1FE 0000 3C10**
  - **Access Size: 8 byte**

Each Timer / Counter Count Register provides the means to load the timer with a preset value on write, and return the current value of the timer on read. The Timer / Counter Count Registers increment once per microsecond.

Table 5-77. **Timer / Counter Count Registers Bit Definitions**

Field	Bits	Description	Type
RESERVED	63:29	Reserved	R0
COUNT	28:0	Value to preset counter on write, and current count value on read	R / W

- **Timer / Counter Limit Registers**
  - **Address: 0x1FE 0000 3C08 and 0x1FE 0000 3C18**
  - **Access Size: 8 bytes**

Each Timer / Counter Limit Register provides the means to enable / disable the interrupt, reload the counter, set the periodic interrupt, and set 'LIMIT' for a counter time-out comparison.

Table 5-78. **Timer / Counter Limit Registers Bit Definitions**

Field	Bits	Description	Type
RESERVED	63:32	Reserved	R0
INT_EN	31	Enable interrupt from this timer. Reset to '0' at power up.	R / W
RELOAD	30	Writes to LIMIT register with this bit set causes the counter to restart at 0x0. Reads as '0'.	W
PERIODIC	29	When set, causes counter to reset to 0x0 when LIMIT is reached.	R / W
LIMIT	28:0	Counter interrupt comparison value	R / W

5.5.1.10 Performance Monitor Registers

In order to gather useful statistics on the performance of the U2S, a pair of registers provide counts of key events. There are only two counters present. The control register selects the input for each of the counters.

Table 5-79. **Physical Address of Performance Monitor Registers**

Register	Physical Address	Access Size
Performance Monitor Control Register	0x1FE 0000 0100	8 bytes
Performance Counter Register	0x1FE 0000 0108	8 bytes

- **Performance Monitor Control Register**
  - **Address: 0x1FE 0000 0100**
  - **Access Size: 8 bytes**

The Performance Monitor Control Register controls the events to be monitored by the Performance Counter Register. The event counters in the Performance Counter Register will be reset when the respective CLR{0,1}bits are set to '1'.

Table 5-80. **Performance Monitor Register Bit Definitions**

Field	Bits	Description	Type
RESERVED	63:16	Reserved	R0
CLR1	15	Clears counter 1	W
RESERVED	14:12	Reserved	R0
SEL1	11:8	Select event source for counter 1. Selected source counter is cleared when CLR1 field is written.	R / W
CLR0	7	Clears counter 0.	W
RESERVED	6:4	Reserved	R0
SEL0	3:0	Select event source for counter 0. Selected source counter is cleared when CLR0 field is written.	R / W

Table 5-81, below, defines the code to select monitored events in the U2S.

Table 5-81. **Performance Counter Event Sources**

SEL0, SEL1	Event Sources
0x0	Number of streaming DVMA read transfers
0x1	Number of streaming DVMA write transfers
0x2	Number of consistent DVMA read transfers
0x3	Number of consistent DVMA write transfers
0x4	Number of TLB misses
0x5	Number of Streaming Buffer read misses
0x6	Number of SBus cycles SBus is granted to DVMA
0x7	Number of bytes transferred using DVMA
0x8	Number of Interrupts
0x9	Number of Interrupt NACK on UPA
0xA	Number of PIO read transfers
0XB	Number of PIO write transfers
0XC	Number of SBus reruns
0XD	Number of SBus cycles SBus is consumed by PIO
0XE - 0XF	Reserved. Counter value is undefined when these sources are chosen.

- **Performance Counter Register**
  - **Address: 0x1FE 0000 0108**
  - **Access Size: 8 bytes**

The Performance Counter Register is a 64-bit read-only register. The value read back contains the counts of two events selected by the Performance Monitor Control Register. The two counters operate independently. Values for both counters are sampled on the same cycle. When the counter reaches its maximum count, it will wrap around to 0x0 and continues counting. Software needs to be able to detect and handle this overflow condition.

Table 5-82. **Performance Counter Register Bit Definitions**

Field	Bits	Description	Type
CNT0	63:32	Contains value for event counter 0	R
CNT1	31:0	Contains value for event counter 1	R

5.5.2 MACIO #1 Registers

Listed in *Table 5-83* below are the registers contained in the MACIO #1. This listing is provided only for reference. For detailed information on the meaning of the registers and any access restrictions, please refer to the 'SBus I/O Chipset Data Manual'.

Table 5-83. **MACIO #1 Registers**

Register	Physical Address	Type	Access Size
DMA2 ESP Registers			
DMA2 Internal ID Register	0x1FF E800 0000	R	32 bits
Control / Status Register	0x1FF E840 0000	R / W	32 bits
Address Register	0x1FF E840 0004	R / W	32 bits
Byte Count Register	0x1FF E840 0008	R / W	24 bits
Test Control / Status Register	0x1FF E840 000C	R / W	32 bits
DMA2 Ethernet Registers			
Control / Status Register	0x1FF E840 0010	R / W	32 bits
Test Control / Status Register	0x1FF E840 0014	R / W	32 bits
Cache Valid Bits Register	0x1FF E840 0018	R / W	32 bits
Base Address Register	0x1FF E840 001C	R / W	8 bits
DMA2 Parallel Port Registers			
Control / Status Register	0x1FF EC80 0000	R / W	32 bits
Address Register	0x1FF EC80 0004	R / W	32 bits
Byte Count Register	0x1FF EC80 0008	R / W	32 bits
Test Control / Status Register	0x1FF EC80 000C	R / W	32 bits
Hardware Configuration Register	0x1FF EC80 0010	R / W	16 bits
Operation Configuration Register	0x1FF EC80 0012	R / W	16 bits
Parallel Data Register	0x1FF EC80 0014	R / W	8 bits
Transfer Control Register	0x1FF EC80 0015	R / W	8 bits
Output Register	0x1FF EC80 0016	R / W	8 bits

Table 5-83. **MACIO #1 Registers** (Continued)

Register	Physical Address	Type	Access Size
Input Register	0x1FF EC80 0017	R / W	8 bits
Interrupt Control Register	0x1FF EC80 0018	R / W	16 bits
SCSI Controller Registers			
Transfer Count Low (7:0) Register	0x1FF E880 0000	R / W	8 bits
Transfer Count Middle (15:8) Register	0x1FF E880 0004	R / W	8 bits
FIFO Data Register	0x1FF E880 0008	R / W	8 bits
Command Register	0x1FF E880 000C	R / W	8 bits
Status Register	0x1FF E880 0010	R	8 bits
Select / Reselect Bus ID Register	0x1FF E880 0010	W	8 bits
Interrupt Register	0x1FF E880 0014	R	8 bits
Select / Reselect Time-out Register	0x1FF E880 0014	W	8 bits
Sequence Step Register	0x1FF E880 0018	R	8 bits
Synchronous Transfer Period Register	0x1FF E880 0018	W	8 bits
FIFO Flags Register	0x1FF E880 001C	R	8 bits
Synchronous Offset Register	0x1FF E880 001C	W	8 bits
Configuration #1 Register	0x1FF E880 0020	R / W	8 bits
Clock Conversion Factor Register	0x1FF E880 0024	W	8 bits
Test (chip Test Use Only) Register	0x1FF E880 0028	W	8 bits
Configuration #2 Register	0x1FF E880 002C	R / W	8 bits
Configuration #3 Register	0x1FF E880 0030	R / W	8 bits
Transfer Count High (23:16) Register	0x1FF E880 0038	R / W	8 bits
Ethernet Controller Registers			
Register Data Port Register	0x1FF E8C0 0000	R / W	16 bits
Register Address Port Register	0x1FF E8C0 0002	R / W	16 bits

5.5.3 MACIO #2 Registers

Listed in *Table 5-83* below are the registers contained in the MACIO #2. This listing is provided only for reference. For detailed information on the meaning of the registers and any access restrictions, please refer to the 'SBus I/O Chipset Data Manual'.

Table 5-84. **MACIO #2 Registers**

Register	Physical Address	Type	Access Size
DMA2 ESP Registers			
DMA2 Internal ID Register	0x1FF 3900 0000	R	32 bits
Control / Status Register	0x1FF 3920 0000	R / W	32 bits

Table 5-84. **MACIO #2 Registers**

Register	Physical Address	Type	Access Size
Address Register	0x1FF 3920 0004	R / W	32 bits
Byte Count Register	0x1FF 3920 0008	R / W	24 bits
Test Control / Status Register	0x1FF 3920 000C	R / W	32 bits
DMA2 Ethernet Registers			
Control / Status Register	0x1FF 3920 0010	R / W	32 bits
Test Control / Status Register	0x1FF 3920 0014	R / W	32 bits
Cache Valid Bits Register	0x1FF 3920 0018	R / W	32 bits
Base Address Register	0x1FF 3920 001C	R / W	8 bits
SCSI Controller Registers			
Transfer Count Low (7:0) Register	0x1FF 3940 0000	R / W	8 bits
Transfer Count Middle (15:8) Register	0x1FF 3940 0004	R / W	8 bits
FIFO Data Register	0x1FF 3940 0008	R / W	8 bits
Command Register	0x1FF 3940 000C	R / W	8 bits
Status Register	0x1FF 3940 0010	R	8 bits
Select / Reselect Bus ID Register	0x1FF 3940 0010	W	8 bits
Interrupt Register	0x1FF 3940 0014	R	8 bits
Select / Reselect Time-out Register	0x1FF 3940 0014	W	8 bits
Sequence Step Register	0x1FF 3940 0018	R	8 bits
Synchronous Transfer Period Register	0x1FF 3940 0018	W	8 bits
FIFO Flags Register	0x1FF 3940 001C	R	8 bits
Synchronous Offset Register	0x1FF 3940 001C	W	8 bits
Configuration #1 Register	0x1FF 3940 0020	R / W	8 bits
Clock Conversion Factor Register	0x1FF 3940 0024	W	8 bits
Test (chip Test Use Only) Register	0x1FF 3940 0028	W	8 bits
Configuration #2 Register	0x1FF 3940 002C	R / W	8 bits
Configuration #3 Register	0x1FF 3940 0030	R / W	8 bits
Transfer Count High (23:16) Register	0x1FF 3940 0038	R / W	8 bits
Ethernet Controller Registers			
Register Data Port Register	0x1FF 3960 0000	R / W	16 bits
Register Address Port Register	0x1FF 3960 0002	R / W	16 bits

5.5.4 SLAVIO Registers

Listed below, in *Table 5-85*, are the registers contained in the SLAVIO. This listing is provided only for reference. For detailed information on the meaning of the registers and any access restrictions, please refer to the 'SBus I/0 Chipset Data Manual'.

Note — The SBus Configuration Register bits 4 through 1 must be set to 0 for this device.

Table 5-85. **Physical Address of SLAVIO Registers**

Register	Physical Address	Type	Access Size
Boot PROM[a]	0x1FF F000 0000 0x1FF F0FF FFFF	R	8 / 16 / 32 bits
Keyboard / Mouse / Serial Registers			
Mouse Control Port Register	0x1FF F100 0000	R / W	8 bits
Mouse Data Port Register	0x1FF F100 0002	R / W	8 bits
Keyboard Control Port Register	0x1FF F100 0004	R / W	8 bits
Keyboard Data Port Register	0x1FF F100 0006	R / W	8 bits
TTYB Control Port Register	0x1FF F110 0000	R / W	8 bits
TTYB Data Port Register	0x1FF F110 0002	R / W	8 bits
TTYA Control Port Register	0x1FF F110 0004	R / W	8 bits
TTYA Data Port Register	0x1FF F110 0006	R / W	8 bits
TOD / NVRAM	0x1FF F120 0000 0x1FF F12F FFFF	R / W	8 / 16 / 32 bits
Generic Port[b]	0x1FF F130 0000 0x1FF F13F FFFF	R / W	8 bits
Floppy Controller Registers			
Digital Output Register	0x1FF F140 0002	R / W	8 bits
Main Status Register	0x1FF F140 0004	R	8 bits
Datarate Select Register	0x1FF F140 0004	W	8 bits
FIFO Register	0x1FF F140 0005	R / W	8 bits
Reserved	0x1FF F140 0006	R	8 bits
Digital Input Register	0x1FF F140 0007	R	8 bits
Configuration Control Register	0x1FF F140 0007	W	8 bits
89C105 Configuration Register	0x1FF F180 0000	R / W	8 bits
Auxiliary I/O Registers			
Aux 1 Register[c]	0x1FF F190 0000	R / W	8 bits
Aux 2 Register[d]	0x1FF F191 0000	R / W	8 bits
Diagnostic Message Register	0x1FF F1A0 0000	R / W	8 bits

Table 5-85. **Physical Address of SLAVIO Registers** (Continued)

Register	Physical Address	Type	Access Size
Modem Register	0x1FF F1B0 0000	R / W	8 bits
Timer / Counter Registers[e]	0x1FF F1D0 0000 0x1FF F1DF FFFF		
Interrupt Controller Registers			
Processor Interrupt Pending Register[e]	0x1FF F1E0 0000		
Processor Clear Pending Pseudo Register[e]	0x1FF F1E0 0004		
Processor Set Soft Interrupt Pseudo Register[e]	0x1FF F1E0 0008		
System Interrupt Pending Register	0x1FF F1E1 0000	R	32 bits
Interrupt Target Mask Register	0x1FF F1E1 0004	R	32 bits
Interrupt Target Mask Clear Pseudo Register	0x1FF F1E1 0008	W	32 bits
Interrupt Target Mask Set Pseudo Register	0x1FF F1E1 000C	W	32 bits
Interrupt Target Register	0x1FF F1E1 0010	R0	32 bits
System Control / System Register	0x1FF F1F0 0000	R / W	32 bits

a. Access to the PROM can be in sizes different that those specified. These define what the bus and device are capable of responding to. Larger size transfers are supported through the U2S by resizing operations before they go out on the SBus.

b. Refer to EBus Devices for description of specific registers.

c. See LED/ FLoppy (SLAVIO Aux 1) Register for a description of this register.

d. See Power Off (SLAVIO Aux 2) Register for a description of this register.

e. These registers are not supported on the SBus Reference Platform systems. **Do not use.**

5.5.5 Audio (APC) Registers

Listed in *Table 5-86* below are the registers contained in the Audio (APC). This listing is provided only for reference. For detailed information on the meaning of the registers and any access restrictions, please refer to the 'SBus I/O Chipset Data Manual'.

Table 5-86. **Physical Address of Audio (APC) Registers**

Register	Physical Address	Type	Access Size
Power Management Registers			
Idle Register	0x1FF DA00 0000	R / W	8 bits
Fan Control Register (Used to Read Clock Frequency)	0x1FF DA00 0020	R / W	8 bits
Unused Register	0x1FF DA00 0024	R / W	8 bits
Generic Bit Ports Register	0x1FF DA00 0030	R / W	8 bits
Audio DMA Registers			
Codec Address Port Register	0x1FF DC00 0000	R / W	8 bits
Codec Data Port Register	0x1FF DC00 0004	R / W	8 bits

Table 5-86. **Physical Address of Audio (APC) Registers** (Continued)

Register	Physical Address	Type	Access Size
Codec Status Port Register	0x1FF DC00 0008	R / W	8 bits
Codec PIO Data Port Register	0x1FF DC00 000C	R / W	8 bits
Control / Status CSR Register	0x1FF DC00 0010	R / W	32 bits
Reserved	0x1FF DC00 0018	W as 0	
Capture Virtual Address Register	0x1FF DC00 0020	R / W	30 bits
Capture Word Count Register	0x1FF DC00 0024	R / W	12 bits
Capture Next Virtual Address Register	0x1FF DC00 0028	R / W	30 bits
Capture Next Word Count Register	0x1FF DC00 002C	R / W	12 bits
Playback Virtual Address Register	0x1FF DC00 0030	R / W	30 bits
Playback Word Count Register	0x1FF DC00 0034	R / W	12 bits
Playback Next Virtual Address Register	0x1FF DC00 0038	R / W	30 bits
Playback Next Word Count Register	0x1FF DC00 003C	R / W	12 bits

5.5.5.1 Fan Control Register

The Fan Control Register is used to read the CPU Frequency. It is defined as follows:

Figure 5-5. **Fan Control Register Bit Definition**

The follow table describes the settings of the Fan Control Register.

Table 5-87. **Fan Control Register Settings**

BR0	BR1	BR2	CPU Speed
0	0	0	128.0 MHz
0	0	1	143.0 MHz
0	1	0	153.9 MHz
0	1	1	167.0 MHz
1	0	0	181.8 MHz
1	0	1	200.0 MHz
1	1	0	222.0 MHz
1	1	1	250.0 MHz

5.5.6 EBus Devices

The EBus provides the following functions:

- access to the USC
- frequency margining control
- lab console control
- writable mapping for the PROM for use with flash devices.

Figure 5-6 below shows a diagram of the EBus devices.

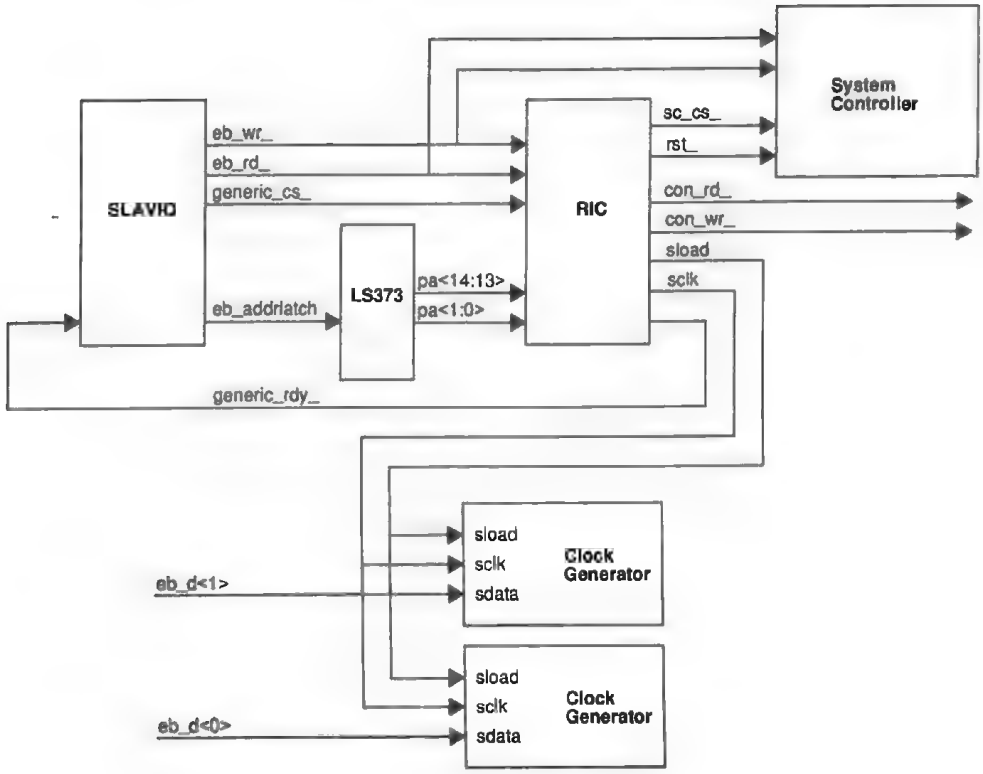


Figure 5-6. **EBus Devices Block Diagram**

Note — All registers described in this section are 8-bit registers.

Table 5-88. **Physical Address of Miscellaneous I/O Registers**

Register	Physical Address
System Controller Address Register[a]	0x1FF F130 0000
System Controller Data Register	0x1FF F130 0004
Writable PROM Address	0x 1FF F138 0000 - 0x1FF F13F FFFF
Frequency Margining Serial Load Register	0x1FF F130 4001
Frequency Margining Serial Load and Reset Register	0x1FF F130 4000

Table 5-88. **Physical Address of Miscellaneous I/O Registers**

Register	Physical Address
Frequency Margining Serial Data In Register	0x1FF F130 4002
SLAVIO Aux 1 Register	0x1FF F190 0000

a. See SC_Address Register for a description of this register.

5.5.6.1 Frequency Margining Registers

- **Frequency Margining Serial Data In Register**
  - **Address:** 1FF.F130.4002

Writing to the Frequency Margining Serial Data In Register will pulse the SCLK input of the UPA and SBus clock generators. Data written to this port will be shifted into the SDATA input of the clock generators. This is a write-only register, and reading from this register yields unpredictable data.

Table 5-89. **Frequency Margining Serial Data In Register Bit Definitions**

Field	Bits	Pupstate	Description	Type
Data	7:0	X	0 = SDATA of UPA clock generator. 1 = SDATA of SBus clock generator.	W

- **Frequency Margining Serial Load Register**
  - **Address:** 1FF.F130.4001

Writing to the Frequency Margining Serial Load Register will generate a pulse to the clock generator and cause the clock frequency to change according to the data previously shifted into the clock generator. The clock generator for UPA and SBus are affected by the load operation. This is a write-only register. Reading from this register yields unpredictable data.

Table 5-90. **Frequency Margining Serial Load Register Bit Definitions**

Field	Bits	Pupstate	Description	Type
Data	7:0	X	Data is ignored.	W

- **Frequency Margining Serial Load and Reset Register**
  - **Address:** 1FF.F130.4000

Writing to the Frequency Margining Serial Load and Reset Register will generate a pulse to the clock generator and cause the clock frequency to change according to the data previously shifted into the clock generator. It also causes the system to reset. The system will be running at the new frequency after reset is completed. This is a write-only register. Reading from this register will yield unpredictable data.

Table 5-91. **Frequency Margining Serial Load and Reset Register Bit Definitions**

Field	Bits	Pupstate	Description	Type
Data	7:0	X	Data is ignored.	W

5.5.6.2 Frequency Setting

The frequency of the main system clock and that of the SBus clock are generated by programmable frequency synthesizers. The system will default to a frequency where the machine is guaranteed to operate. The default frequency must be a function of the most significant bit of the frequency code output by the RIC chip in order to insure the proper minimum operating frequency for the ASIC PLLs. Software must read the frequency code from the Fan Control Register of the APC (address 0x1FF DA00 0020, bits <3:1>) and then use the serial programming path to set the frequency to the desired value. For more information on the APC register refer to the APC chip specification.

Rather than having only one table, software might consider using a linked list of pointers to tables with one field of the list being the CPU revision. This would permit reuse / redefinition of frequency codes over time, so if the cpu speed curve improves, it will be possible to upgrade the flash ROM to take advantage of faster CPU modules thereby improving the performance of the machine.

Note — SBus Reference Platform systems board will default to the slower speed. Software will adjust the speed if it determines that the board is capable of running faster.

5.5.7 External Registers

5.5.7.1 Status Register

- **Address:** 0x1FF.F1C0.0000
- **Status:** Read-Only

This register contains the revision number of the USP-1. The register description is given below.

Table 5-92. **Status Register Bit Definition**

31	30	29	28	37	26	25	24	23
X	X	X	X	X	REV3	REV2	REV1	REV0

Table 5-93. **Status Register Bit Meanings**

Status Register Bits	Bit Setting	Description
REV3, REV2, REV1, REV0	0	Engineering Version (Preliminary)
REV3, REV2, REV1, REV0	1 - E	Released Version #
REV3, REV2, REV1, REV0	F	Temporary Version

5.5.7.2 LED Register

- **Address:** 0x1FF.F1C0.0004
- **Status:** Read/Write

This register drives the four user LEDs on the USP-1.

- A '0' indicates the LED is ON.
- A '1' indicates the LED is OFF.

Table 5-94. **LED Register Bit Definition**

31	30	29	28	37	26	25	24	23
X	X	LED3	LED2	LED1	LED0	X	X	X

5.5.7.3 User Switch Register

- **Address:** 0x1FF.F1C0.0008
- **Status:** Read-Only

This register reads the setting of the user switch.

- A '0' indicates the switch is ON.
- A '1' indicates the switch is 'OFF'

Table 5-95. **User Switch Register Bit Definition**

31	30	29	28	37	26	25	24	23
X	X	SW3	SW2	SW1	SW0	X	X	X

5.5.7.4 RESET DS1620 Register

- **Address:** 0x1FF.F1C0.000C
- **Status:** Write-Only

This register controls the assertion of RESET to the DS1620 Thermostat:

- Writing a '0' to this register causes the RESET input of DS1620 to be asserted.
- Writing a '1' to this register causes the RESET input of DS1620 to be de-asserted.

The Default value of RST (after a reset) is 0.

DS1620 reset is done manually to satisfy the 10 millisecond pulse width requirement imposed after DS1620 writes to its non-volatile memory. For more information refer to the Dallas Semiconductor System Extension Data Book.

Table 5-96. **RESET DS1620 Register Bit Definition**

31	30	29	28	37	26	25	24	23
X	X	SW3	SW2	SW1	SW0	X	X	X

5.5.7.5 Command Register

- **Address:** 0x1FF.F1C0.0020
- **Status:** Read/Write

This register enables the over-temperature shutdown and maps SBus Slots 0 and 1 to either the SBus or the VMEbus.

Table 5-97. **Command Register Bit Definitions**

31	30	29	28	37	26	25	24	23
X	X	ESHUT	X	X	X	X	PTHRU1	PTHRU0

Table 5-98. **Command Register Bit Meanings**

Status Register Bits	Bit Setting	Description
ESHUT	0	Over-temperature shutdown enabled (Default)
	1	Over-temperature shutdown disabled
PTHRU(x)	0	SBus Slot (x) is reserved for SBus (Default)
	1	SBus Slot (x) is reserved for VMEbus

5.5.7.6 DS1620 Serial Data Register

- **Address:** 0x1FF.F1C0.0010
- **Status:** Read/Write

This register is used to transmit and receive serial data from the DS1620 Thermostat.

- DQ indicates DS1620 Serial I/O.

Table 5-99. **DS1620 Serial Data Register Bit Definitions**

31	30	29	28	37	26	25	24	23
X	X	X	X	X	X	X	X	DQ

6

Interrupts

6.1 Overview

Interrupts in the USP-1 make use of the UPA provided interrupt vector mechanism. Simply stated, all interrupts are delivered to the processors through a packet write which provides 24 bytes of data to the processor (three double registers). Level sensitive software acknowledged interrupts, which would formerly be communicated through dedicated interrupt wires, are converted into interrupt packets and delivered to the processor.

Hardware interrupt sources guarantee that the upper 53 bits of the first interrupt word as well as the last two 64 bit words are 0. The least significant 11 bits of the first word contain an interrupt number (INR) which indicates the type of interrupting event. Software uses the INR to index into a table which will typically supply the IRL, PC of the interrupt service routine, and the ARGs for the routine.

The UPA permits devices which are slave only to deliver interrupts either by performing master interrupt transactions, or through a single wire which informs the "system interconnect" to deliver the interrupt transaction.

Hardware guarantees that all registers associated with servicing a device interrupt are programmable from within that device's address space. Mapping registers may reside in the interrupting device's address space, or within some other agent's address space. No central resource is defined in the global address space to handle the functions which are abstractly delegated to the "system interconnect".

Two types of interrupt lines go into the concentrator: 'pulse' and 'level'. The distinction between these is not software visible, but is explained for clarity.

Processing hardware treats the two types of interrupts slightly differently. In the case of the 'level' interrupt, the concentrator takes the set of asserted 'level' interrupt lines, scans them and sends the code corresponding to that interrupt once per scan time. Hardware within the U2S detects the first assertion of a code, and causes a state transition which queues up an interrupt packet for some target UPA device. A three state state-machine is used. This state machine will transmit only one interrupt as long as it remains in the pending state regardless of how many interrupt codes it gets for a source. Writing the pending register causes a transition to the idle state and "re-arms" the state machine to accept another interrupt. Refer to Section 6.6.1, "Interrupt States" for a discussion of the state transitions.

'Pulse' interrupts are scanned and delivered to U2S in a similar fashion; however, only one code is given per 'pulse'. The distinction is subtle, but very important. In the case of the existing interrupts, multiple interrupt sources can contribute to the physical line signalling the interrupt, but there is no restriction which guarantees

that software knows the interrupt line has properly deasserted. In the case of 'pulse' interrupts, this is required. There must be the equivalent of the pending register in the device sourcing the interrupt. Writing this register guarantees that the interrupt line has deasserted and therefore pulsed. As a consequence, the state machine in the U2S corresponding to a 'pulse' interrupt has only two states. Refer to Section 6.6.1, "Interrupt States" for a discussion of the state transitions.

6.2 Interrupt Initialization

All fields in all mapping registers listed above reset to 0. When the valid bit is cleared, no interrupts will be generated from that interrupt group. Prior to receiving the first interrupt, software must program all mapping registers to set INR, and all UPA_TID registers. All interrupt state registers should be cleared back to idle. Should software wish to change the UPA_TID for a group, it can either change the TID without synchronization (i.e., there is no guarantee of which target the interrupt will go to which target for some minor period) or can perform the following to guarantee proper logical sequencing:

1. Disable the target ID mapping register by setting V=0.
2. Follow the write with a read to the same address to confirm that the write has made it out to the U2S.
3. Re-program the TID to the desired value.
4. Re-enable the interrupt group by setting V=1 (can be combined with the write in step 3).

As an alternative, software can change the TID for an interrupt source while servicing an interrupt for that source. Hardware guarantees that any transaction not in progress when the valid bit is disabled will not proceed. Once the valid bit is enabled again, interrupts will proceed. This points out that the valid bit ONLY gates delivery of interrupts to the processors. It does not affect other state transitions within the interrupt logic. This implies that an interrupt can be delivered immediately upon first setting valid bit if an interrupt condition exists.

6.3 Interrupt Servicing

Upon receipt of a *mondo vector*[1], and assuming that PSTATE.IE=1, the processor will take a type 0x60 trap. The INR is used to index into a table which provides three pieces of information: the IRL, the PC for the interrupt service routine, and the args which need to be supplied. A softint is issued to call the interrupt service enqueue routine with this information.

1. *Mondo vector*: An interrupt request packet containing three 64-bit fields: PC, DATA (PTR), and DATA.
   PC: Program Counter for the interrupt service routine.
   DATA: two data fields associated with the interrupt. These can be defined to include the address of the transaction that erred, the contents of a status register, etc. -U2P does NOT send any data with interrupts.
   For more information refer to the Interrupts Section in the Sun4u System Architecture manual.

Note — Currently UltraSPARC-1 uses PSTATE.IE bit to enable the reception of interrupt packet and the generation of trap for IRL <4:0>. Software should not disable PSTATE.IE for a long period of time when servicing IRL <4:0>.

When the interrupt service routine has performed all device level servicing, it will call an operating system service to de-queue the service routine. This OS service WST write the clear interrupt register for the appropriate interrupt source in order to re enable interrupts from that source. Information on the appropriate clear interrupt register should be saved at the time of enqueue.

6.4 Interrupt Sources

Interrupts in the systems come from I/O devices, system error conditions, and software. Sources of I/O device interrupts are SBus slots, the graphics interface, and MACIO/FEPS/SLAVIO. All I/O device interrupts are connected to the Interrupt Concentrator, the RIC chip. The Interrupt Concentrator scans through its inputs and encodes the interrupt into 6-bits for U2S. U2S maintains state information on all of the interrupt sources and sends an interrupt packet to the proper processor.

Each interrupt signal connected to the Interrupt Concentrator can be assigned with a unique interrupt number. The interrupt number allows the software to identify the source of interrupt without polling the devices. With the exception of serial ports and keyboard / mouse, devices in the system do not share interrupts.

Excluding SBus interrupts, all other interrupts can be targeted to any UPA device. The interrupt target ID for the interrupt is specified in its Interrupt Mapping Register. SBus interrupts on the same slot share the same target ID.

Cross-calls are the interrupt generated by one processor to another processor. They are sent through the UPA interconnect. SOFT Interrupts are also generated by the software, but the interrupt is directed to the processor itself.

Note — MACIO #2 interrupts come in through SBus slot 3.

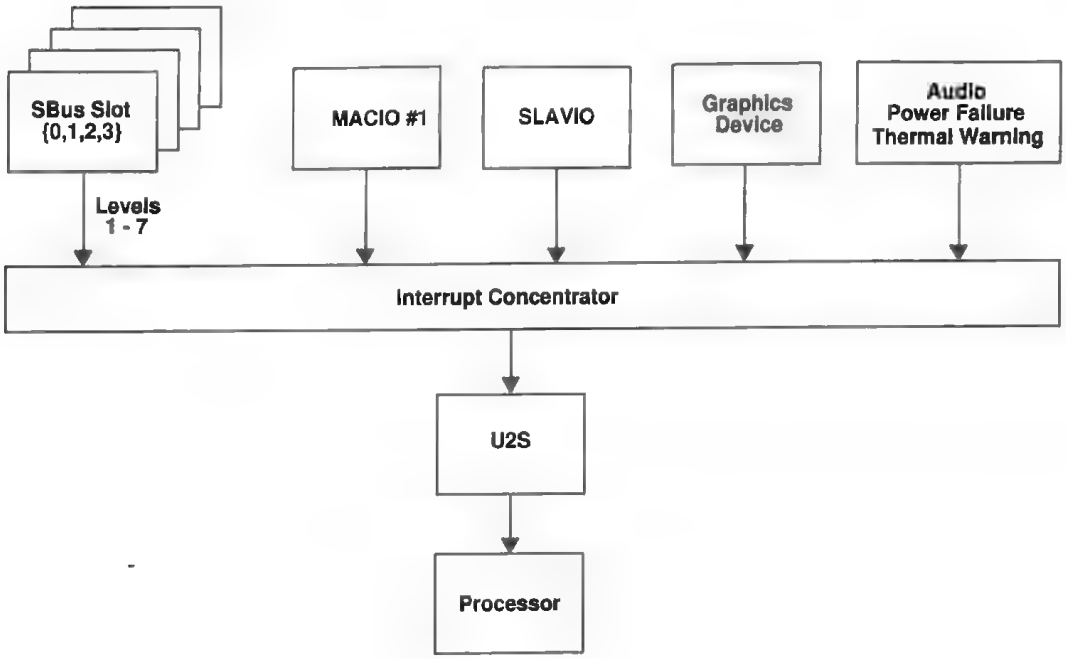


Figure 6-1. **Interrupt Block Diagram**

6.4.1 SBus Interrupts

SBus provides seven levels of interrupts. The specification allows interrupts be bussed among slots or connected radially to the system interrupt controller. The SBus Reference Platform provides radial connections from interrupt sources to the Interrupt Concentrator. This enables interrupt control logic to send a unique interrupt number for each interrupt source.

SBus prioritizes the interrupt with level 7 being the highest and level 1 the lowest. U2S will dispatch the interrupt based on the priority as assigned by SBus. The SBus interrupt level a SBus card uses is specific to the device. So is the interrupt control registers for interrupt enable and interrupt clear. Please consult documentation for the SBus cards.

6.4.2 Timer Counter Interrupts

There are two internal timer / counters in U2S. Each one is capable of generating periodic or one-shot interrupt. The two timer / counters operate independently of each other. When the counter reaches its limit as specified in the Limit Register and the interrupt enable bit is set, an interrupt will be generated. The interrupt is self-cleared after processor receives the interrupt. No explicit operation is needed to clear the interrupt. For more detailed description of Timer / Counters please refer to Chapter 5, The Programmer's Visible State.

6.4.3 On-board Device Interrupts

MACIO and SLAVIO are used to provide Sunness functionality in the system. Interrupts sourced from FEPS/ MACIO are Ethernet, SCSI, and Parallel Port. Please refer to the MACIO specification for more details on Ethernet, SCSI and Parallel Port interrupts.

SLAVIO generates interrupts on behalf of Serial Ports, Keyboard / Mouse and Floppy Controller. SLAVIO interrupts are different from the rest of interrupt sources in the following ways:

1. **Interrupt Sharing:** Interrupts from Keyboard / Mouse and Serial Ports are shared in the device. The software will have to poll SLAVIO internal register to find out which device caused the interrupt.

2. **Interrupt Prioritizing:** Two interrupts are controlled by SLAVIO, Serial Ports / Keyboard / Mouse and Floppy Controller interrupts. SLAVIO prioritizes interrupt internally to the chip. If both interrupts are active internally, the Floppy Controller interrupt will not be seen until the Serial Ports / Keyboard / Mouse interrupt is cleared from the source. Software will most likely make use of a higher level service routine which should poll the interrupt register in SLAVIO and issue interrupts to the proper lower level service routine for floppy or Serial Ports / Keyboard / Mouse.

All external interrupt sources to SLAVIO will be disabled. Software must make sure to disable all internal interrupt sources other than KMS and floppy. For more detailed information on Serial Port / Keyboard / Mouse and Floppy Controller interrupts please refer to the SLAVIO Specification.

6.4.4 Error Interrupts

Internal errors detected by U2S will be reported through interrupts. Error related information will be recorded in U2S internal registers. The following errors are reported through interrupts:

1. PIO Read - SBus late error.

2. PIO Write - UPA UE, UPA CE, SBus late error, SBus error ack, SBus time-out.

3. DMA Read / Write UE and CE.

6.4.5 Cross-calls

Processors can send interrupts to each other through the UPA interface. The interrupt is delivered as a UPA interrupt packet. The sequence of sending cross-calls in the system is described below. Please refer to UltraSPARC Programmer's Reference Manual and Sun5 System Architecture Manual for detailed descriptions.

1. Load information into Interrupt Vector Data registers. These are three 8-byte registers in UltraSPARC. The first 8 bytes contains the PC of the cross-call. The remaining 2 registers contain interrupt specific information.

2. Write to Interrupt Vector Dispatch register to send the interrupt. The target processor is specified through virtual address.

3. Check the interrupt status through Interrupt Vector Dispatch Status register. If the NACK bit is set, the software has to restart the send sequence from beginning. If the BUSY bit in this register is set, software should wait until it is cleared before a new interrupt is sent.

6.4.6 Software Interrupts

The processor can send an interrupt to itself by setting bits in the SoftInt Register in the UltraSPARC. No interconnect transactions are generated in this case.

6.5 Interrupt Concentrator

The Interrupt Concentrator logic is implemented in the RIC chip to encode interrupts from various sources into a 6-bit code that the U2S uses to identify the sources of interrupt. The value for the graphics interrupt is shown in *Table 6-1* below.

Table 6-1. **Interrupt State Transition Table**

INT Code	Interrupt Source
100011	Graphics Interrupt
100110	Spare edge sensitive interrupt

The Interrupt Concentrator scans the interrupt inputs in fixed order. If there is no active interrupt, the IDLE code will be sent to U2S. When it detects an active interrupt, the Interrupt Concentrator will change the code from IDLE to one of the active codes. It can deliver one interrupt code to U2S every SBus clock cycle with an initial latency of 3 clock cycles. If multiple interrupts are active at the same time, the interrupts behind the current one will observe the latency by the Interrupt Concentrator. The worst case latency introduced by Interrupt Concentrator is 50 SBus clocks. This only describes the latency from the assertion of an interrupt line to receipt of the interrupt code in the U2S. The time to dispatch the mondo vector to the target processor is less than 20 UPA cycles.

The Interrupt Concentrator does not keep track of any state for level interrupts. For pulse interrupts, it keeps track of the assertion of the interrupt, and transmits only one code for each assertion. Filter logic within the chip inhibits sending additional codes to U2S until the signal has deasserted.

Level interrupt codes will be sent to U2S whenever there is active interrupt present. U2S needs to remember that an interrupt has already detected and know to ignore incoming interrupt code.

6.6 U2S Interrupt Handling

6.6.1 Interrupt States

Interrupts generated by I/O devices are level or pulse sensitive. They are converted into UPA interrupt packets. U2S needs to keep track of the state of each level interrupt to avoid re-sending the same interrupt that is already received by the processor.

For the three state state-machine, the interrupt states are IDLE, XMIT, and PEND. The two state state-machine used for the pulse interrupts requires only the idle and transmit states. Following table shows the state transition of interrupt.

Table 6-2. **Interrupt State Transition Table**

State Transition	Description
IDLE -> XMIT	An active interrupt is detected from Interrupt Concentrator.
XMIT -> PEND	The interrupt has been delivered to the processor. This transition is present only for the three state version.
XMIT -> IDLE	The interrupt has been delivered to the processor. This transition is present only for the two state version.
PEND -> IDLE	The interrupt has been cleared by software.

Note — The PEND state is used to indicate that the interrupt has already been sent to the processor and the interrupt is not yet cleared. This is to provide software with information on what has happened to the interrupt. For the state machine to transition to this state, the valid bit in the mapping register must be set. Interrupts for which the valid bit is not set can transition to the XMIT state, but may not dispatch to the UPA target.

The interrupt state information can be obtained from Interrupt State Registers in U2S. Two bits in each register define the state of a interrupt.

6.6.2 Interrupt Prioritizing

If there is only one interrupt in the XMIT state, U2S will dispatch the interrupt to its target processor if the processor is available to receive a interrupt packet. If there are multiple interrupts in the XMIT state, the dispatching of interrupt is based on fixed priority. Among interrupts with same priority, round-robin priority is used. The following table shows the priority of U2S generated interrupts with 8 being the highest priority and 1 being the lowest priority.

Table 6-3. **Interrupt Dispatching**

Priority	Interrupt Source
1	SBus Slot 0 L1, SBus Slot 1 L1, SBus Slot2 L1, SBus Slot 3 L1, Power Management Wakeup Interrupt.
2	SBus Slot 0 L2, SBus Slot 1 L2, SBus Slot2 L2, SBus Slot 3 L2, Parallel Port Interrupt and Thermal Warning.
3	SBus Slot 0 L3, SBus Slot 1 L3, SBus Slot2 L3, SBus Slot 3 L3, SCSI Interrupt and Ethernet Interrupt
4	SBus Slot 0 L4, SBus Slot 1 L4, SBus Slot2 L4, SBus Slot 3 L4, Keyboard Interrupt and Mouse Interrupt
5	SBus Slot 0 L5, SBus Slot 1 L5, SBus Slot2 L5, SBus Slot 3 L5, Graphics Interrupt and UPA64S Interrupt.
6	SBus Slot 0 L6, SBus Slot 1 L6, SBus Slot2 L6, SBus Slot 3 L6, Timer 0 Interrupt and Timer 1 Interrupt.

Table 6-3. **Interrupt Dispatching** (Continued)

Priority	Interrupt Source
7	SBus Slot 0 L7, SBus Slot 1 L7, SBus Slot2 L7, SBus Slot 3 L7, Keyboard / Mouse / Serial Combined Interrupt and Serial Interrupt.
8	ECC UE error Interrupt, ECC CE error Interrupt, SBus error Interrupt, Power Fail Interrupt, Audio Interrupt, and Floppy Interrupt.

6.6.3 Interrupt Dispatching

U2S maintains a lookup table for interrupts dispatched by U2S. The table contains target ID and interrupt number. Their functionality is described in Section 5.5.1.8, Interrupt State Registers.

U2S implements one interrupt dispatch buffer internally. The dispatch buffer is not software visible. It contains information about the interrupt that is about to be dispatched or has already been dispatched but not received by the processor. U2S will dispatch a new interrupt only when the dispatch buffer is empty. Once the interrupt is in the dispatch buffer, it will be delivered to the UPA device.

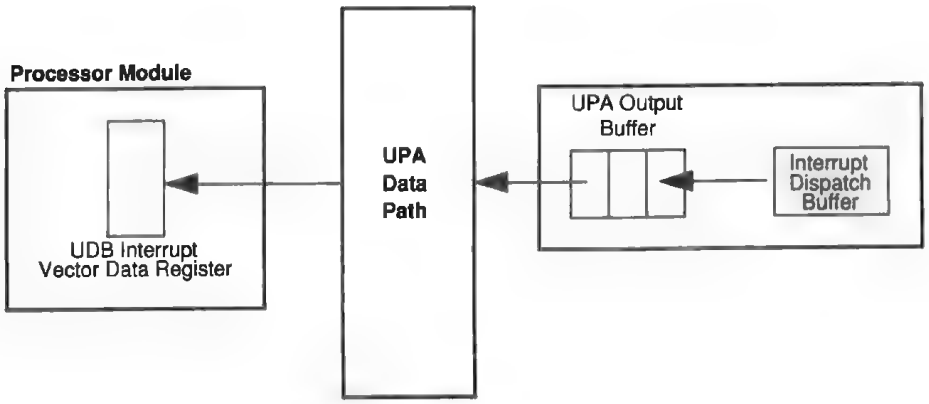


Figure 6-2. **UPA Interrupt Dispatch Block Diagram**

The Interrupt Vector Data Registers in UltraSPARC UDB chips are used to store the interrupt data delivered by U2S or other UPA devices. There is one set of registers provided by each UltraSPARC module. After the data is loaded into Interrupt Vector Data Registers, the module will not be able to receive any incoming interrupt packet until the registers are emptied. The "Busy" bit in the UltraSPARC Interrupt Vector Receive Register will be set when the interrupt data is loaded into Interrupt Vector Data Registers. This bit can be cleared by software.

If a UPA device attempts to send an interrupt while the registers are full, System Controller will respond to the interrupt requester with an NACK. The "NACK" response informs the requester that the interrupt request is not successful and it should try to send it again.

After the processor module receives the interrupt packet, an interrupt trap will be generated if the IE bit of the UltraSPARC PSTATE Register is set to 1. The trap type for the interrupt trap is 0x60. Please refer to UltraSPARC Programmer's Reference manual for more details.

6.7 UPA Slave Interrupt Bindings

UPA slave interrupts present an interesting technical problem for the Software I/O framework. There is both an interrupter and an interrupt proxy. The interrupt proxy is the device which masters the interrupt transaction to the system interconnect. In the cases of the SBus systems, the U2S performs this function.

The problem to solve has to do with the difference in address space for the interrupt device and the interrupt proxy. Registers in the address space of the interrupt proxy are necessary to the function of the interrupter yet the interrupter and proxy do not have to share a common device tree node other than root.

This presents no problem during normal interrupt servicing since the only registers which need to be written are in the interrupter. That is why the pulse interrupts make use of the two state state-machine rather than the three state state-machine. However, interrupt mapping or remapping are made difficult because of the lack of a common address space. During mapping, the boot firmware or OS code has to be able to algorithmically determine the address of the interrupt proxy given the address of the interrupter.

To permit algorithmic determination of the address of the interrupt proxy, it is required that the addresses of interrupter and interrupt proxy be related. Two cases are considered: one interrupter per interrupt proxy, and up to 3 interrupters per interrupt proxy.

6.7.1 One Interrupter Case

The first case occurs when there is one interrupter per interrupt proxy. In this case, the UPN (UPA Port Number) will be even for the interrupter and the corresponding proxy will be at the next larger odd port number.

Let UPN[4:0] represent the port number of the interrupter and let $UPN_p[4:0]$ represent the port number of the interrupt proxy.

- The interrupter will have a port number such that $UPN_I[0] == 0$.
- The interrupt proxy will have a port number such that $UPN_P[0] == 1$.
- The interrupter and proxy relationship will be $UPN_I[4:1] == UPN_p[4:1]$.

6.7.2 Two or Three Interrupter Case

The other supported case involves having two, or possibly three, interrupters per interrupt proxy. Current desktop hardware has hooks in place for up to two interrupters per interrupt proxy.

Again, let $UPN_I[4:0]$ represent the port number of the interrupter(s) and let $UPN_P[4:0]$ represent the port number of the interrupt proxy.

- The interrupter(s) will have a port number such that $UPN_I[1:0] == 2, 1, \text{or } 0$.
- The interrupt proxy will have a port number such that $UPN_P[1:0] == 3$.
- The interrupter and proxy relationship will be $UPN_I[4:2] == UPN_P[4:2]$.

Software should allocate the interrupt mapping registers such that the first mapping register goes with the interrupter with $UPN_I[1:0] == 2$ and the second goes with the interrupter with $UPN_I[1:0] == 1$. This will help maintain some compatibility between the single and multiple interrupter cases.

6.7.3 Compatibility Issues

It should be clear from the discussion above that the case of the single interrupter per interrupt proxy and the case of multiple interrupters per interrupt proxy are similar, but not compatible. The case where $UPN_I[1:0] == 0$ illustrates the conflict. The single interrupter per proxy case will have $UPN_I[1:0] == 1$ whereas the multiple interrupter per proxy case will have $UPN_I[1:0] == 3$.

Software should use the knowledge of whether it is a single or multiple interrupter per proxy to determine the binding between port numbers and the location of the mapping registers within the proxy address space.

7

SBus IOMMU

7.1 Overview

SBus IOMMU performs virtual address to physical address translation during DVMA cycles. SBus master devices provide a 32-bit virtual address at the beginning of DVMA transfers. IOMMU translates it into a 41 bit physical address.

The IOMMU consists of a TLB, a TSB, and a Software managed data structure (the PTE). Sixteen entries of fully associative TLB is implemented in U2S ASIC. TSB is a second level lookup table which resides in memory. Hardware performs TSB lookup when the translation cannot be found in the TLB. An error is returned to the device if TSB lookup fails to locate a valid mapping. The software managed data structure provides information during the setup of TSB table entry.

The SBus IOMMU supports two different page sizes, 8 Kbyte and 64 Kbyte. Mixed page sizes can be used in the system, but the TSB table lookup only assumes the smaller page size. No overlapping of pages is allowed. Bypass operation is supported to allow a device which has its own translation facility to bypass IOMMU.

7.2 Mode of Operations

Depending on the value of 'MMU_EN' bit of IOMMU Control Register, 'BY' bit of SBus Slot Configuration Register and SBus virtual address bits <31:30>, the IOMMU is put into different operating modes as shown in *Figure 7-1* below.

Table 7-1. **IOMMU Mode of Operation**

MMU_EN	BY	VA <31:30>	Mode
X	1	00	Bypass
X	1	01	Slot Configuration Register access
0	1	1X	Pass-through
1	1	1X	Translation
0	0	XX	Pass-through
1	0	XX	Translation

7.2.1 Translation Mode

IOMMU is in translation mode if translation is enabled, MMU_EN bit set, and the device performing transfers is not using bypass mode. When translation is started, IOMMU hardware will perform TLB lookup first. If TLB a miss happens, hardware automatically starts TSB lookup. If TSB lookup locates a valid mapping for the virtual page, information in the TSB entry will be loaded into TLB and translation continues. If TSB lookup results in a miss, an error will be returned to the SBus master.

The virtual address consists of two fields, virtual page number and page offset. The page offset stays the same from virtual address to physical address. SBus IOMMU supports two page sizes, 8 Kbyte and 64 Kbyte. The conversion of virtual address to physical address is shown in *Figure 7-1* below.

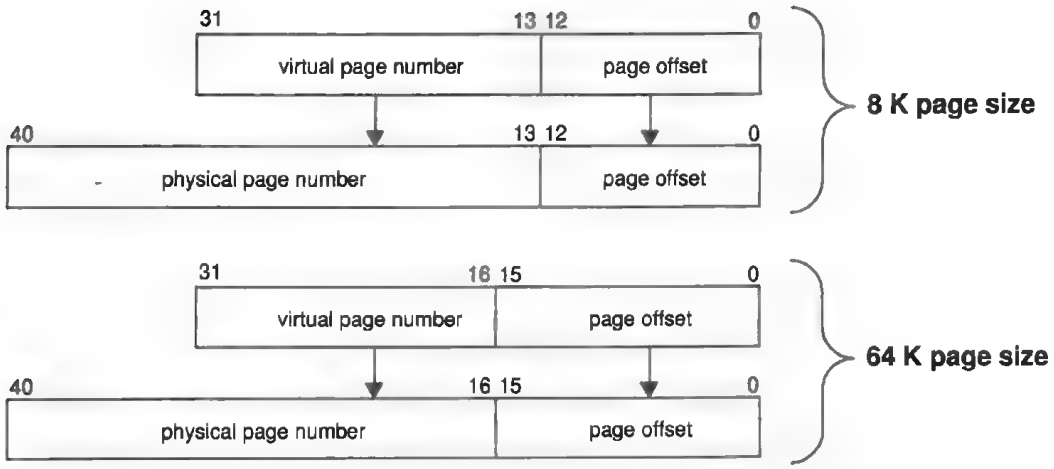


Figure 7-1. **Virtual Address to Physical Address Translations**

7.2.2 Bypass Mode

The implementation of IOMMU allows the SBus device to have its own MMU and bypass the IOMMU supported by the system. A device is operating in bypass mode if the 'BY' bit in its Slot Configuration Register is set and the high order two virtual address bits, VA <31:30>, are set to '00'.

Although device in bypass mode has the access to full physical address space, it provides only 30 bits of address on the SBus. To access the 41-bit physical address space of the system, the high order address bits, PA <40:30>, come from the SBus Slot Configuration Register (SSCR) SEGA <40:30>. The physical address is formed by concatenating SEGA <40:30> and VA <29:00>. If the DMA access is crossing 1 Gbyte boundary, the device needs to change the SEGA <40:30> to point to the next 1 Gbyte segment. The SEGA <40:30> is writable by device through DVMA access as shown in Table 5-1: IOMMU Mode of operation.

A device in the 41-bit of physical address space can be either cacheable or non-cacheable. The "CP" bit of the SSCR controls whether a coherent transaction should be issued to the UPA interconnect. In SBus Reference Platform systems only the memory space is cacheable. Software should set the "CP" bit properly in order to avoid misbehavior of the hardware.

7.2.3 Pass-through Mode

When the IOMMU is disabled and the DVMA transfer is not in bypass mode or accessing the SSCR, the access is considered to be in pass-through mode. Pass-through mode allows access to the lower 2 or 4 Gbyte of memory address space only. The high order 10 or 9 bits of the physical address will be padded with 0. A DVMA access in pass-through mode will always cause a coherent transaction to the UPA interconnect.

7.3 Translation Storage Buffer

Translation Storage Buffer (TSB) is a translation table in memory. It contains mapping information for the virtual pages. IOMMU hardware will look up this table if a translation cannot be found in the TLB. Each entry in the table takes 8 bytes.

Several TSB table sizes are supported in the system. The size of TSB table is specified by the TSB-SIZE field of the IOMMU Control Register. Supported table sizes are 1 KByte, 2 KByte, 4 KByte, 8 KByte, 16 KByte, 32 KByte, 64 KByte, and 128 KByte entries. Software must set up TSB before it allows translation to start.

7.3.1 Translation Table Entry

Each entry in the TSB table is called a Translation Table Entry (TTE). The entry contains translation information for a virtual page. The IOMMU hardware reads in translation information from TTE during a TLB miss. The "VALID" bit in the entry must be set to contain valid information. Information stored in the TTE is shown in *Table 7-2* below.

Table 7-2. **Translation Table Entry Data Format**

Field	Bits	Description
DATA_V	63	Valid bit, indicating the TTE entry has valid mapping
RESERVED	62	Reserved
DATA_SIZE	61	Page size of the mapping 0 = 8 Kbyte 1 = 64 Kbyte
STREAM	60	Set if the page is streammable. Please refer to Chapter 8, Streaming Buffer for more description of the Stream Mode.
LOCAL_BUS	59	Set if the physical address points are local to the SBus.
DATA_SOFT_2	58:51	Assigned for software use.
RESERVED	50:41	Reserved
DATA_PA	40:13	Contains bits <40:13> of physical address, some of the low order bits are not used for 64 Kbyte page.
DATA_SOFT	12:7	Assigned for software use.
RESERVED	6:5	Reserved; read as 0
CACHEABLE	4	Set if the page is mapped cacheable
RESERVED	3:2	Reserved; read as 0

Table 7-2. **Translation Table Entry Data Format** (Continued)

Field	Bits	Description
DATA_W	1	Writable page; Attempts to write to a page when this bit is set to '0' will result in an error ack
RESERVED	0	Reserved; read as 0

7.3.2 TSB Lookup

During the TSB lookup physical address for the TTE entry is formed based on the following information:

- Base address of the TSB table.
- Assumed page size during TSB lookup, TBW_SIZE of the IOMMU Control Register.
- Size of TSB table.

TSB Base Address Register contains the physic address of the first TTE entry in the TSB table. The table must be aligned on an 8 Kbyte boundary regardless of table size. Lower order 13 bits of this register are assumed to be 0. The physical address for entry in TSB table is formed by adding the base address and the offset together.

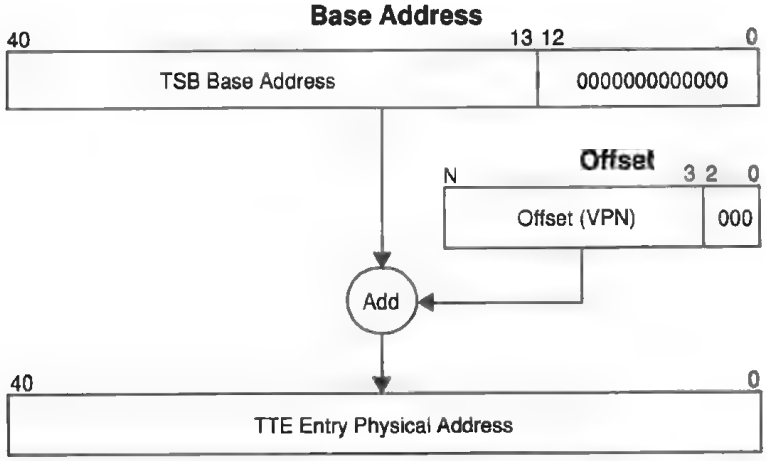


Figure 7-2. **Computation of TTE Entry Address**

Offset of the TSB table is calculated based on a direct mapped TSB table. *Table 7-3* shows how the offset is obtained. The lower order 3 bits is set to 0x0 because each entry is 8 bytes in size.

Table 7-3. **Offset to TSB Table**

TSB Table Size	N	Offset (8 Kbyte TSB lookup page size) TBW_SIZE = 0	Offset (64 Kbyte TSB lookup page size) TBW_SIZE = 1
1 Kbyte	12	VA <22:13>, 000	VA <25:16>, 000
2 Kbyte	13	VA <23:13>, 000	VA <26:16>, 000
4 Kbyte	14	VA <24:13>, 000	VA <27:16>, 000
8 Kbyte	15	VA <25:13>, 000	VA <28:16>, 000

Table 7-3. **Offset to TSB Table**

TSB Table Size	N[a]	Offset (8 Kbyte TSB lookup page size) TBW_SIZE = 0	Offset (64 Kbyte TSB lookup page size) TBW_SIZE = 1
16 Kbyte	16	VA <26:13>, 000	VA <29:16>, 000
32 Kbyte	17	VA <27:13>, 000	VA <30:16>, 000
64 Kbyte	18	VA <28:13>, 000	VA <21:16>, 000
128 Kbyte	19	VA <29:13>, 000	Not allowed

a. Refer to *Figure 7-2* for the definition of 'N'.

TBW_SIZE should be set to '0' if either an 8 Kbyte page size or a mixed 8 Kbyte and 64 Kbyte page sizes are used for DVMA mappings. Each 64 Kbyte page will use up 8 entries of TTE if the TBW_SIZE is set to '0'. Software must fill all 8 entries with the same information if mixed page size is used.

7.4 Translation Errors

Errors detected by the IOMMU during translation will be reported to the SBus master device. It is up to the SBus device to report the error. Information associated with the erroneous transaction should also be kept in the SBus device. No system resources are allocated for this purpose. Errors detected by IOMMU are invalid errors and protection errors. An invalid error happens if the TTE read by the IOMMU hardware is an invalid entry. A protection error happens when a device is trying to write to a page that has no write permission. Both errors are reported through SBus error acknowledge.

7.5 IOMMU Demap

After the mapping between the virtual address and the physical address is established, any change to the mapping information needs to demap the existing mapping before new mapping can be used by the device. Demap is required for the following occasions:

- taking down existing mapping to make physical memory available to other virtual addresses;
- changing access permission to a page.

During IOMMU demap, the SBus device is not allowed to use the page being demapped. If a device tries to access a page that is being demapped, unexpected results may happen. Following events are needed to demap a page in the IOMMU.

- Flush the streaming cache if the page is marked streammable. Software is required to wait until notification that the flush has completed before moving on.
- Update proper TSB entry with new information.
- Perform TLB flush with virtual page number.

TLB flush is initiated by writing to IOMMU Flush Address Register with specified virtual page number. Match criteria are different for 8 Kbyte and 64 Kbyte page sizes. Hardware performing the flush will adjust match criteria based on the page size. The matched entry in the TLB will be marked invalid.

7.6 TLB Initialization and Diagnostics

IOMMU provides direct access to its internal resources, such as TLB Tag, TLB Data, LRU Queue, and Match Comparison Logic. Please read Section 5.4, System Controller Registers, for more detailed information.

After power is turned on, the contents of IOMMU is undefined. Before any DVMA is allowed to use the IOMMU, all TLB entries need to be invalidated. This is done by writing '0' to 'V' bit in every entry of TLB Data RAM.

8

Streaming Buffer

8.1 Overview

The Streaming buffer implemented in U2S is a small size fully associative cache managed by hardware to accelerate SBus DVMA to and from memory. A DVMA page can be mapped in consistent or stream mode. Only the page mapped streammable is allowed to use the Streaming Buffer.

Stream DVMA needs software maintenance. These include page invalidation, page flush, and flush synchronization. Certain restrictions are necessary to ensure the correctness of data and proper performance of the Streaming Buffer.

8.2 Consistent DVMA and Stream DVMA

DVMA to and from memory can be operated in stream mode or consistent mode. The operating mode is controlled by the "ST" bit of the IOMMU mapping. Since the stream property is controlled solely by the IOMMU mapping, the property is assigned based on page. It's not possible to have stream and consistent mode assigned to a page at the same time. The stream and consistent property can be changed from time to time, but software needs to take proper action during demap. This will be described in a later section.

8.2.1 Consistent DVMA

Consistent DVMA participates in cache coherence and does not make use of the streaming buffer. It is useful for IOPBs and cases where the overhead of managing the streaming buffer outweighs the benefits. Consistent DVMA are used during the following DVMA transfers:

- DVMA transfer to and from memory mapped in consistent mode.
- DVMA transfer to and from memory using IOMMU bypass or pass through mode.
- DVMA transfer to and from non-cacheable UPA address space.

8.2.1.1 Consistent DVMA Write

DVMA writes to memory in consistent mode will be observed in issuing order at the coherence domain. Write buffers are provided in the U2S for consistent write operation. No synchronization mechanism is needed for consistent DVMA write buffers because interrupt is guaranteed not to surpass the previous consistent write to the UPA interconnect and write ordering is maintained. No order of completion is guaranteed between consistent DVMA write and the following operations:

- Steam write
- SBus DVMA to SBus device
- PIO accesses.

All consistent DVMA writes will be observed at the UPA interconnect. Since the unit of coherency at UPA interconnect is 64 bytes and U2S does not implement cache at the UPA domain, any consistent DVMA partial write of less than 64 bytes needs to be performed with the following operations in order:

1. Ownership read from coherence domain.
2. Merge write data with returned data.
3. Write data into memory.

Once the U2S gets the temporary ownership of the cache line, any access to the same cache line will be blocked by the System Controller until the memory write is completed. DVMA partial write to memory in consistent mode is discouraged because of the expensive operations involved.

8.2.1.2 Consistent DVMA Read

During consistent DVMA read the U2S takes a snapshot of the data from coherence domain and returns the data to the SBus device. U2S will issue a Read-To-Discard UPA transaction to get the data without transfer of ownership. The read will always be 64 bytes in size. Only requested data will be returned to the SBus device. Remaining unused data will be thrown away.

Data returned to the SBus master will become stale if a UPA master updates the same memory locations with new value. To get the latest data the SBus device needs to perform another DVMA read.

8.2.2 Stream DVMA

U2S provides sixteen 64-byte lines of read-ahead and post-write cache to speed up stream mode DVMA. The allocation of cache line is based on one line per virtual page of 8 Kbytes. Although IOMMU can map a page to be 8 Kbytes or 64 Kbytes, the Streaming Buffer will assume smaller size mapping. The allocation is managed by hardware using LRU algorithm.

Data transferred in stream mode may not be observed in issuing order. It also needs software intervention to help forcing data into coherence domain. Only DVMA access to and from system memory is allowed to use stream mode.

8.2.2.1 Stream Read

When SBus device starts a DVMA read, U2S will use SBus virtual address to look up Streaming Buffer and the IOMMU at the same time. If the access results in a hit on the cache line, data will be provided directly from the Streaming Buffer. Otherwise a line will be allocated to the page, if not already done, and data will be fetched from the coherence domain and filled into the Streaming Buffer. This saves the latency and reduces the consumption of UPA memory bandwidth of having to get data from coherence domain on every DVMA read.

To further enhance performance, the Streaming Buffer will perform prefetch when it senses current DVMA operation is going to consume the last byte of data in the cache line. However if the address is going to cross a page boundary, prefetch is inhibited. The prefetch capability is transparent to software.

Once the data gets into the Streaming Buffer it will not remain coherent with the rest of the system. Data can be overwritten at the coherence domain without the notice of the Streaming Buffer and therefore DVMA master. Software needs to make sure pages marked streammable are not sensitive to this characteristic.

8.2.2.2 Stream Write

The Streaming Buffer acts as an accumulating write buffer for the DVMA write. DVMA writes with sequentially increasing address will be accumulated in the cache line. When the write reaches the end of a cache line, a flush operation will be initiated. Doing this saves the expensive operation of a DVMA partial write to memory.

The side-effect of buffering write is that the data may stay in the Streaming Buffer for a long period of time if no software mechanism exists to force the data out. Flush and flush synchronization are needed to ensure that the consumer of the data gets the most updated information.

8.3 Streaming Buffer Management

To ensure data consistency across the system, software needs to manage the Streaming Buffer. Operations managing the Streaming Buffer include invalidation, flush, and flush synchronization. Invalidation and flush are done by the same command. It involves a PIO write to the Streaming Buffer Page Invalidate / Flush Register with virtual page number provided as PIO write data. The entry with the matching virtual page number will be invalidated if the page is clean or flushed/invalidated if the page contains dirty data. Eight invalidate or flush operations are needed for a 64 Kbyte page since the tagging occurs only 8 Kbyte boundaries and it is possible to have all 8 pages present in the cache.

8.3.1 Streaming Buffer Invalidation

The Streaming Buffer needs to be invalidated in the following scenarios.

- **Before a device starts its first DVMA read:** A cache line may be brought into the Streaming Buffer long before the device starts the DVMA read. If the memory location is updated after the line is in the cache, the device is not able to see the updated data.

- **When a page is demapped:** The Streaming Buffer also stores mapping information to speed up the flush operation. The mapping information is also used during prefetch. When a page is demapped, the mapping information in the Streaming Buffer needs to be invalidated also.

8.3.2 Streaming Buffer Flush

A Streaming Buffer Flush can be triggered by any of the several sources listed below. The first four in the list are invisible to the software.

- **End of line flush:** When a DVMA write reaches the end of a cache line, the hardware will perform a line flush if the flush buffer is available. Otherwise, the line will stay in the cache until it is bumped out.
- **Non-sequential write to the same line:** Any non-sequential write to a cache line will cause the existing line to be flushed before new data can be accepted.
- **Line eviction on the same page:** If a line is partially filled and the device starts writing to a new line on the same page, buffered data for the previous write has to be flushed before new data can be accepted.
- **Line eviction different page:** This happens when all the cache lines are used up and from the incoming DVMA accesses a page with no line allocated to it. If the LRU line has dirty data, it needs to be flushed before the line can be allocated to a new page.
- **Software triggered flush:** Software needs to flush the Streaming Buffer at the end of DVMA transfers. A software initiated flush will never take longer than 0.5 seconds in a properly working system. A time-out of this duration can be used to recover from some undetectable hang.

To make sure all previous flush operations are completed at the coherence domain, U2S provides a mechanism to synchronize the flush operation. The flush synchronization involves a PIO write to the Streaming Buffer Flush Synchronization Register with the physical address of the flush flag provided as PIO write data. Only one write of the synchronization register is required as a barrier for all previous flush / invalidate writes.

Following is a sequence of events in performing a flush and synchronization

1. Grab the lock of the flush flag.
2. Initialize the flush flag to 0x0 (in memory).
3. Flush dirty pages.
4. Synchronize flushes.
5. Poll the flush flag (located at the PA contained in the Flush Synchronization Register) until it becomes 0xl.
6. Release the lock of flush flag.

8.4 Streaming Buffer Error Handling

There is no difference in reporting stream DVMA and consistent DVMA errors. Interested readers should consult the Chapter 9, Error Handling, for more details on error reporting. The Streaming Buffer does not keep any error state and therefore does not need software intervention for error management.

8.5 Software Notes

Certain limitations are imposed on the Streaming Buffer. Some of them are for performance reasons and some may cause misbehavior of the system. These limitations are listed below.

- To avoid unnecessary flushes a DVMA write should use an increasing sequential address. Using a decreasing sequential address will cause a flush operation on every DVMA write. Jumping around lines on the same page or jumping back and forth on the same line will increase the frequency of flushes on a DVMA write and unnecessary prefetch during a DVMA read. All of these actions should be avoided.
- Avoid address wrapping on DMA streaming writes.
- To amortize the overhead of a flush it is recommended that a stream DVMA be used to transfer a large amount of data instead of small chunks of data.
- Whenever possible, a larger burst size should be used to maximize performance.
- The Streaming Buffer is tagged with a virtual address. It does not provide alias detection on the mappings. Two devices may have different virtual pages that are mapped in the same physical page. If both are writing to the same physical memory location, or one device is reading and the other is writing to the same memory location, the results of the operations are not guaranteed.
- If the same physical page is allocated to two different DVMA devices and their physical address space is not overlapped, aliasing can be used to avoid unnecessary flushes due to interleaving accesses between two different lines on the same page.

9

Error Handling

9.1 Overview

This chapter describes the error detection / correction and error reporting mechanisms supported in SBus Reference Platform systems.

Errors detected in the system are classified into the two types: fatal hardware errors and non-fatal hardware errors. Fatal errors may result in a system reset if the error reset is enabled by software. Actions taken on non-fatal errors include interrupts, status registers accesses, and no action.

9.2 Error Detection and Reporting

SBus Reference Platform provides error detection at interconnect levels and inside each ASIC. Parity and ECC are used to protect the interconnects at various levels. Following diagram shows where and how the interconnects are protected in SBus Reference Platform systems.

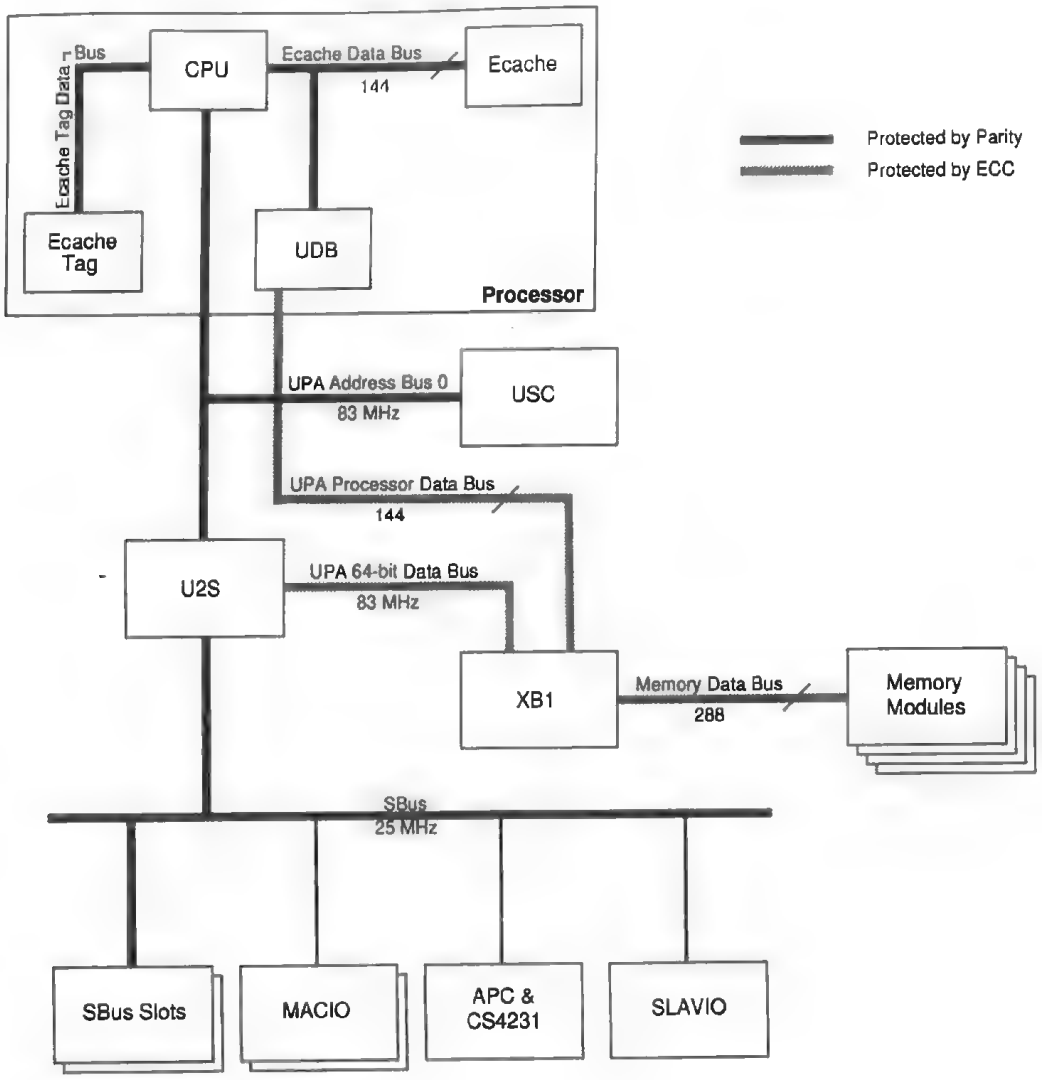


Figure 9-1. **USP-1 Error Detection / Correction Block Diagram**

9.2.1 Fatal Hardware Errors

Fatal errors detected by the hardware are UPA Address Parity Error, USC Master Queue Overflow, and E-cache Tag Parity Errors.

The UPA address bus carries address and control information for UPA transactions. The format of the address packet is described in Section 3.3.1, UPA Interconnect, of the Hardware Manual. Two address busses are supported in the SBus Reference Platform system. One address bus connects between the U2S and USC processors and the other bus connects between the USC and UPA graphic devices. Only the address bus connecting the U2S and USC processors supports parity. The address bus between the USC and UPA graphic devices does not support parity. The parity bit is driven by a master UPA port at the same time the address packet is driven to the bus. The parity bit will be set to '1' when there is an even number of '1s' on the

address bus. When the address packet is received by the USC parity checking is performed. If the address packet is targeted to another UPA slave port that has no parity error, the USC will route the address packet to the target device on the other address bus.

9.2.1.1 Address Parity Error

An address parity error is considered a fatal error to the system. Both the UPA slave port and the USC can detect an address parity error. The action taken on parity errors is described below.

- **Address parity error detected by a UPA slave port:** The device detecting the address parity error should send a P_FERR reply to the USC if it is enabled to do so. The USC will generate a reset to the system upon receipt of the P_FERR. The USC will log the fatal error condition in the USC Port Status Register and the USC Control Register.
- **Address parity error detected by the USC:** The USC generates a reset to the system and logs the error condition in the USC Control Register.

Software should clear the address parity error bit by writing a '1' to it. Failing to do so may cause confusion to the software if another address parity error happens after the existing one.

9.2.1.2 Master Queue Overflow

The USC maintains a queue for each UPA master class. The maximum depth of the USC master queue for each master class is defined in the USC Port Status Registers. If a UPA master device issues more transactions than the size of the UPA master queue, an overflow condition exists. This can happen due to a malfunction of the hardware or software a setting a wrong value in the UPA master's UPA Configuration Register. The USC will reset the system if an overflow condition exists. It also sets the overflow status bit in the USC Port Status Register. Software should clear the error bit by writing a '1' to it after recovering from the reset condition.

9.2.1.3 E-cache Tag Parity Error

The datapath between the processor and the E-cache Tag RAM is also protected by parity. Information stored in the E-cache Tag RAM includes 22 bits of E-cache tag, 3 bits of E-cache state, and 4 bits of parity. The UltraSPARC is the only device performing parity generation during tag update and parity checking during tag read. There are 4 parity bits for 25 bits of data. The relationship of parity bits and their protected data fields is listed below:

- Parity <0>: E-cache Tag <7:0>
- Parity <1>: E-cache Tag <15:8>
- Parity <2>: E-cache Tag <21:16>
- Parity <3>: E-cache Tag <2:0>.

The trap will be taken by UltraSPARC if it detects an E-cache tag parity error. If the access is caused by an UPA snoop operation, a P_FERR P_REPLY is generated to the UPA resulting in a system reset.

9.2.2 Non-fatal Hardware Errors

9.2.2.1 E-cache Data Parity Error

The E-cache data bus connects between the UltraSPARC processor, the E-cache data RAM and the UltraSPARC Data Buffer (UDB). The 128-bit wide data bus is protected by byte parity. Bad parity on this bus can be caused by faulty devices or interconnects or an UE detected by the UDB. Both UltraSPARC and UDB perform parity generation and parity checking.

The processor performs parity checking on:

- Processor read from E-cache, UDB, and system resources.
- Processor write to UDB or devices on the UPA interconnect.
- Data provided by E-cache data RAM during copyback and writeback.

A parity error detected during E-cache data access will cause a trap to the UltraSPARC. Detailed action of the UltraSPARC is described in the UltraSPARC Programmer's Reference Manual.

The UDB performs parity checking on:

- Processor write to UPA devices.
- Data provided by E-cache during copyback and writeback.

A parity error on the E-cache Data bus detected by UDB will be converted into an uncorrectable ECC error to the UPA datapath. If the target device is an UPA port, the port detecting the UE should also report the error. If the destination is memory, the UE will stay in memory until a UPA master tries to use the data.

9.2.2.2 UPA Datapath Uncorrectable Error

The UPA datapath implemented in SBus Reference Platform systems consists of Buffered MX (XB1), and a collection of wires connecting between the UltraSPARC UDB, XB1, memory, U2S, and UPA graphic devices. There are three major busses provided for this purpose:

- Memory Data Bus - 256 bits of data and 32 bits of ECC information.
- UPA Data Bus (Processor) - 128 bits of data and 16 bits of ECC information.
- UPA Data Bus (I/O) - 64 bits of data and 8 bits of ECC information.

Devices connected to the UPA datapath perform ECC generation, checking and correction. In the SBus Reference Platform, the devices supporting ECC generation / checking / correction are UDB and U2S.

Two types of ECC errors are checked by the UPA port, Correctable Errors (CE), and Uncorrectable Errors (UE). CE can come from the following sources:

- Corruption on the datapath, such as UPA, and memory
- Faulty device, such as DRAM, XB1, or UPA device.

In addition to normal UE sources there are other cases a device can force a UE to the system. The UE handling software needs to find out what exactly caused the UE in the system. Information related to UltraSPARC induced UE is described in UltraSPARC AFSR/AFAR. Information related to U2S induced UE is described in Section 5.5.1.3, U2S SBus Control Register. Possible sources of UE are listed below:

- Corruption on the datapath, such as UPA, and memory

- Faulty device, such as DRAM, XB1, or UPA device
- E-cache data parity error
- SBus data parity error during DVMA write
- SBus data parity error during PIO read.

If the device detects a CE, data will be corrected before it is used. Data transfer continues as if there is no error. The receiving device will log the error information and report the errors. Upon receiving an UE, the error will be carried over, maybe in a different way, to make sure the erroneous data is not consumed. The receiving device will log the error information and report the error.

U2S detects and corrects ECC errors on the data it receives. The checking and correction are done on the following operations:

- PIO writes to U2S and devices controlled by U2S
- SBus DVMA read from memory or UPA devices
- SBus DVMA write to memory.

U2S reports ECC errors to the processor by interrupt if ECC checking and ECC interrupt are both enabled. Error information is logged in the UE and CE AFSR/AFAR. For more information about the ECC Control Register, UE, and CE Error Registers please read the Chapter 5, The Programmer Visible State.

The other device the performs ECC checking / correction is the UltraSPARC UDB. The UDB checks / corrects ECC errors on processor instruction / data fetch, and interrupt packets.

9.2.2.3 UPA Time-out

UPA time-out can happen during a slave read to a non-existing UPA port or access to an unsupported / nonexisting device controlled by the UPA port, for example reading from an empty SBus slot.

A UPA transaction needs to be completed with a reply in order to avoid hanging the initiating device. There is no bus timer defined in the UPA interconnect to keep track of pending transactions. To avoid hanging the system due to an inadvertent access to a non-existing device, the USC will check for the existence of the device before relaying the access to the targeted port. A UPA port should signal its existence by sending an IDLE reply to the USC when there is no access to it. If the port exists in the system, it is the responsibility of the UPA port to issue a reply to any access targeted to it. Access to non-existent UPA ports are handled differently for slave read and slave write:

- Slave write to a non-existing UPA port will be ignored by the USC. The IADD bit in the USC Port Status Register will be set.
- Slave read from a non-existent port will be terminated with a read time-out reply from the USC and the IADD bit in the USC Port Status Register will be set. The handling of read time-out error is device dependent.
- PIO read from a non-existent or faulty SBus device will also result in a UPA time-out error report to the initiating UPA master.

9.2.2.4 UPA Read Error

Other than address parity errors the UPA does not provide an error reply to write transactions. The USC or UPA port can only send error replies to read transactions. Write errors detected by the UPA slave port should be reported through an interrupt. A UPA slave can send the UPA read error reply for various reasons not to

be described here. For example, the U2S will respond with a read error reply if an SBus error acknowledge is received from a SBus slave device. The USC will return an error reply to the master port if an error reply is received from a UPA slave, or a UPA master attempts to perform a non-cached UPA transaction to memory space. In the latter case the IADD bit of the Port Status Register will be set to reflect the error.

9.2.2.5 SBus Parity Error

The SBus provides parity protection on its datapath. A single bit parity is provided for 32-bit data in normal transfer mode and 64-bit data in Extended Transfer mode. Odd parity is used for 32-bit/64-bit data wires and DP parity wire.

In order to enable parity checking, devices involved in the data transfer must support parity generation and checking capability. The FCode on the SBus device provides the parity attribute. System software can examine the FCode to find out whether a SBus device supports parity or not. The way to enable parity checking on the SBus device is device dependent. It is not described in this document.

SBus Slot Configuration Registers (SSCR) in U2S control the parity checking and generation in U2S during SBus data transfer. One SSCR is provided for each SBus device. Setting the 'PE' bit of this register to '1' will enable the U2S to perform parity checking on transfers to and from the SBus device.

The U2S acting as an SBus controller / master / slave performs parity checking on the following situations, if parity checking is enabled for the device:

- **Translation Phase** when the SBus master device drives the virtual address to D<31:0> of SBus. The U2S will return an SBus error acknowledge to the SBus device. It is up to the device to report the error.
- **Extended Transfer Information Phase** when the SBus master device provides extended transfer information to the U2S. The U2S will return an SBus error acknowledge to the SBus device. It is up to the device to report the error.
- **Data Transfer Phase** and **Extended Data Transfer Phase** when the U2S is receiving data from the device. This can happen on a PIO read from the SBus device or on an SBus DVMA write to the U2S, the UPA, and/or to the memory.

A parity error detected on a PIO read will be reported with uncorrectable data being delivered to the processor and the PIO error bit in the SBus Control Register being set. A parity error detected during an SBus DVMA write will be reported with an SBus late error to the SBus master with proper timing. Action taken by the master on the SBus late error is device dependent. If the DVMA write is targeted to the U2S, the write will be ignored. If it is targeted for a UPA device or memory the U2S will provide the data with an UE and set the DMA_PERR status bits in the U2S SBus Control Register. Error handling software needs to clear the bits after servicing the UE.

An SBus device can also detect a parity error during the Data Transfer Phase or Extended Data Transfer Phase of a PIO write or a DVMA read. The error reporting mechanism is device specific.

9.2.2.6 SBus Time-out

The U2S maintains a timer to keep track of slave accesses to SBus devices. If the device does not exist or exists but does not respond to the access over a period of time, the U2S will terminate the cycle with an SBus error acknowledge when the timer expires. The timer starts ticking when the SBus slave cycle starts with an SBus Address Strobe. The time-out period is defined by the SBus specification to be 256 SBus clock ticks.

An SBus time-out on PIO access is reported differently depending on the type of access. A PIO read terminated by an SBus time-out will be reported with a UPA time-out reply to the initiating UPA port. An SBus write time-out will be reported through an interrupt. Please refer to the Chapter 5, The Programmer Visible State chapter for the logging of error information.

An SBus time-out can also happen when an SBus master directly accesses the SBus slave device. In this case the handling of the SBus time-out is device dependent and is not described in this document.

9.2.2.7 SBus Error

An SBus slave device can signal an error to master access due to various error conditions. Examples of error conditions detected by the slave devices are unsupported address space, accesses with wrong size, protection error, error conditions internal to the device, etc.

An SBus error during a PIO read will be reported with an UPA error reply to the processor. An SBus error during a PIO write will be reported as an interrupt and the U2S will log the error information in the SBus AFSR/AFAR.

The U2S can also return an error acknowledge to the SBus master during DVMA transfers. The error can be caused by an UE at memory, an UE UPA interconnect, translation error, access to non-existing UPA device, error internal to the UPA device, etc. The reporting of SBus DVMA errors is device dependent.

9.2.2.8 SBus Late Error

The SBus allows slave devices to report an error condition after proper acknowledge is given to the master device. The timing of Sbus late error assertion is defined in the SBus specification. A late error can happen during both PIO and DVMA accesses. A late error during a PIO read/write access will be reported by interrupts and the error information will be logged in the SBus AFSR/AFAR.

The U2S will assert a late error if it detects a parity error during the Extended Data Transfer Phase or during the Data Transfer Phase of DVMA write transfers. Please refer to Section 9.2.2.5, SBus Parity Error, for the details about the action taken by the U2S. The handling of SBus DVMA late errors is device specific.

9.2.2.9 DVMA Errors

The UPA UE Registers log information during some DVMA errors. The list summarizes the error reporting behavior:

1. If an UE interrupt is enabled, an interrupt will be posted when the U2S detects an UE.

2. Data will be ignored by the Streaming Buffer if the UE is caused by a Streaming Buffer prefetch. The entry will stay invalid if an UE is detected.

3. An UE caused by a demanded DVMA read from a streammable page will cause an ERROR ACK to the SBus device even if the UE does not happen on the 8-byte quantity requested by the device. This is to avoid paying one SBus cycle penalty on a read hit, because the error status is not available when the SBus acknowledge is asserted. The Streaming Buffer entry will be set to an invalid state if this happens.

4. An UE caused by a DVMA read from a consistent page will cause an ERROR ACK to the SBus device only if the UE happens on the data requested. An SBus DATA ACT will be provided if the requested bytes do not have an error.

5. If a DVMA partial write transaction totally writes over the 8 bytes getting an UE, the error will be written over. Good data and check bits are provided for the 8 bytes when writing it back to memory. If a DVMA transaction partially overwrites or does not overwrite the data having the UE, the U2S will force the UE to memory.

6. If an SBus parity error is detected during a DVMA write, the U2S will force an UE only to the bytes having SBus parity error. If there is no SBus parity error within the aligned 8-byte data, the good data and check bits will be provided to memory.

7. If an SBus parity error is detected during a PIO read, the U2S will force an UE only to the bytes having the SBus parity error.

9.2.2.10 IOMMU Translation Error

During an SBus DVMA operation, the IOMMU will provide translation to the SBus virtual address. The IOMMU also checks for access violation. Errors detected by the IOMMU are access to an invalid page and access with a protection violation. An invalid error happens when the DVMA virtual page does not have a valid physical page mapped to it. A protection error occurs when the SBus master tries to write to a page that is marked as read-only. Both errors will be reported with an SBus error acknowledge to the device. The actual reporting of the translation errors is device dependent.

9.2.3 Summary of Error Reporting

Table 9-1 summarizes the reporting of fatal errors detected in an SBus Reference Platform system.

Table 9-1. **Summary of Fatal Error Reporting in SBus Reference Platform**

Error Type	Type of Operation	System Action	CPU Action	Error Register
UPA Address Parity Error	All	Reset	Reset	USC Port Status Register USC Control Register
Master Queue Overflow	All	Reset	Reset	USC Port Status Register
E-cache Tag RAM Parity Error	CPU IFetch, CPU LD/ST, UPA Snoop	Reset	Reset	Ultra SPARC AFSR / AFAR

Following table summarizes the reporting of non-fatal errors detected in the SBus Reference Platform system.

Table 9-2. **Summary of Non-fatal Error Reporting in SBus Reference Platform**

Error Type	Type of Operation	Response to UPA[a]	S.F. Trap	Error Register[b]	SBus Action
E-cache Data RAM Parity Error	CPU IFetch CPU LD / ST UPA Snoop	Force bad ECC if UPA Snoop	Yes	UltraSPARC AFSR / AFAR	No
UPA UE	PIO Read Memory Read UPA Interrupt	No	Yes	UltraSPARC AFSR / AFAR[c]	No
	PIO Write	UE Interrupt	No	U2S UE AFSR / AFAR[c]	No SBus cycle
	DVMA Read	UE Interrupt	No	U2S UE AFSR / AFAR[c]	SBus Error Ack[d]
	DVMA Write	UE Interrupt	No	U2S UE AFSR / AFAR[c]	SBus Data Ack
UPA CE	PIO Read Memory Read	No	Yes	Ultra SPARC AFSR / AFAR	No
	PIO Write	CE Interrupt	No	U2S CE AFSR / AFAR	No
	DVMA Read	CE Interrupt	No	U2S CE AFSR / AFAR	SBus Data Ack
	DVMA Write CE	CE Interrupt	No	U2S CE AFSR / AFAR	SBus Data Ack
UPA Read Error UPA Time-out	PIO Read Memory Read	UPA Read Error UPA Time-out	Yes	UltraSPARC AFSR / AFAR USC Port Status Register	No
	DVMA Read	UPA Read Error UPA Time-out	No	No	SBus Error Ack[d]
	DVMA Write	No	No	No	SBus Data Ack
SBus Parity Error	PIO Read	Force bad ECC to UPA	Yes	UltraSPARC AFSR / AFAR USC Port Status Register	SBus Data Ack
	PIO Write	No	No	No	SBus Data / Error Ack[e]
	DVMA Virtual Address	No	No	No	SBus Error Ack[d]
	DVMA Read Data Phase (ETI)	No	No	No	SBus Data Ack[e]
	DVMA Write Data Phase	Force bad ECC to UPA	No	U2S Bus Control Register	SBus Data Ack
SBus Time-out	PIO Read	UPA Time-out	Yes	UltraSPARC AFSR / AFAR	SBus Error Ack
	SBus Time-out	SBus Async Error Interrupt	No	U2S SBus Async Error AFSR / AFAR	SBus Error Ack
	DVMA Read DVMA Write	No	No	No	SBus Error Ack[d]

Table 9-2. **Summary of Non-fatal Error Reporting in SBus Reference Platform**

Error Type	Type of Operation	Response to UPA[a]	S.F. Trap	Error Register[b]	SBus Action
SBus Error Ack	PIO Read	UPA Read Error	Yes	UltraSPARC AFSR / AFAR	SBus Error Ack
	PIO Write	SBus Async Error Interrupt	No	U2S SBus Async Error AFSR / AFAR	SBus Error Ack
	DVMA Read DVMA Write	No	No	No	SBus Error Ack[d]
SBus Late Error	PIO Read	SBus Async Error Interrupt	No	U2S SBus Async Error AFSR / AFAR	SBus Data Ack
	PIO Write	SBus Async Error Interrupt	No	Us2 SBus Async Error AFSR / AFAR	SBus Data Ack
	DVMA Read DVMA Write	No	No	No	SBus Data Ack Late Error
IOMMU Translation Error	DVMA Read DVMA Write	No	No	No	SBus Error Ack[d]

a. This assumes error interrupts are enabled.

b. Information logged in the AFSR / AFAR depends on the type of error and the state of the error status. Please refer to the UltraSPARC Programmer's Reference Manual and Section 5.4.2 System Control Internal Register Definitions for more details.

c. An UE may be caused by other errors in the system. Additional information is available in the UltraSPARC and U2S registers. Please read the Section 5.5.1.4, ECC Registers for more details.

d. The handling of an SBus error acknowledge during a DVMA cycle is device dependent. The device should keep the DVMA from the channel from getting an error ack and post an interrupt to the system.

e. The handling of an SBus parity error detected by the SBus device is device specific. The device should generate an interrupt to the system.

9.3 Unreported Errors

Certain error conditions are not reported by the systems. Examples of these errors are listed below. Please be aware that the list is not intended to enumerate all possibilities.

- A write to a non-existent or disconnected UPA port. The IADD bit in the Port Status Register will be set, but no error is reported.
- A write to a read-only register in the USC or the U2S is ignored.
- A non-cached UPA write to memory. The IADD bit in the UPA Port Status Register will be set but no error is reported.
- A read from a write-only register in the USC or the U2S gets garbage data.
- An UPA bus error or time-out during a DVMA write is ignored.
- Sending an interrupt to a non-existent UPA port or disconnected UPA port. The IPORT bit in the Port Status Register will be set, but no is error reported.

10

UPA Packet Handling

10.1 Introduction

One of the major functions of the USC is the forwarding and handling of UPA packets. This chapter describes in detail how the USC accomplishes this.

10.2 USC Transaction Table

Table 10-1 on page 10-2 shows all of the possible transactions that the USC will handle and all the possible outcomes. Some notes on the table are provided below:

1. MEM_ADDR is an address in the main memory address space.
2. SLV_ADDR is an address in the slave address space of a UPA port that is present in the system.
3. CPU_ADDR is an address in the address space located in the CPU.
4. SYSIO_ADDR is an address in the address space located in the U2C.
5. FFB_ADDR is an address in the address space located in the FFB Graphics (not supported in SBus Reference Platform).
6. ERR_ADDR is an address that is not supported in the system.
7. VAL_ID is a valid interrupt target id.
8. INV_ID is an invalid interrupt target id.
9. IADDR refers to the IADDR bit in the SC_Port_Status register of the initiator.
10. IPORT refers to the IPORT bit in the SC_Port_Status register of the initiator.

Table 10-1. **USC Transaction Types**

Transaction Type	Master ID	Address	P_Request / S_Request	P_Reply	S_Reply to Master	S_Reply to Slave
P_NCRD_REQ	CPU, U2S	SLV_ADDR	P_NCRD_REQ	P_RAS P_RASB P_RERR P_RTO	S_RAS S_RAS S_ERR S_RTP	S_SRS S_SRS - -
		MEM_ADDR	(set IADDR)	-	S_ERR	-
		ERR_ADDR	(set IADDR)	-	S_RTO	-
P_NCBRD_REQ	CPU, U2S	SLV_ADDR	P_NCBRD_REQ	P_RAB P_RASB P_RERR P_RTO	S_RBU S_RBU S_ERR S_RTO	S_SRB S_SRB - -
		MEM_ADDR	(set IADDR)	-	S_ERR	-
		ERR_ADDR	(set IADDR)	-	S_RTO	-
P_NCWR_REQ	CPU, U2S	SLV_ADDR	P_NCWR_REQ	P_WAS	S_WAS	S_SWS
		MEM_ADDR	(set IADDR)	-	S_WAS	-
		ERR_ADDR	(set IADDR)	-	S_WAS	-
P_NCBWR_REQ	CPU, U2S	SLV_ADDR	P_NCBWR_REQ	P_WAB	S_WAB	S_SWB
		MEM_ADDR	(set IADDR)	-	S_WAB	-
		ERR_ADDR	(set IADDR)	-	S_WAB	-
P_INT_REQ	CPU, U2S	VAL_ID	P_INT_REQ	P_IAK	S_WAB	S_SWIB
		INV_ID	(set IPORT)	-	S_WAB	-
(interrupt nacked)		VAL_ID	-	-	S_INAK	-
P_RDS_REQ	CPU	MEM_ADDR	-	-	S_RBU	(from mrm)
		CPU_ADDR	P_RDS_REQ	P_RERRC	S_ERR	-
		SYSIO_ADDR	P_RDS_REQ	P_RERRC	S_ERR	-
		FFB_ADDR	-	-	S_ERR	-
		ERR_ADDR	(set IADDR)	-	S_RTO	-
	U2S	any address	UNDEFINED[a]	-	-	-
P_RDO_REQ	CPU	MEM_ADDR	-	-	S_RBU	(from mem)
	U2S	MEM_ADDR	S_CPI_REQ (to CPU)	P_SACK P_SACKD P_SNACK	S_RBU S_RBU S_RBU	S_CRAB S_CRAB (from mem)
	CPU, U2S	CPU_ADDR	P_RDO_REQ	P_RERRC	S_ERR	-
		SYSIO_ADDR	P_RDO_REQ	P_RERRC	S_ERR	-
		FFB_ADDR	-	-	S_ERR	-

Table 10-1. **USC Transaction Types**

Transaction Type	Master ID	Address	P_Request / S_Request	P_Reply	S_Reply to Master	S_Reply to Slave
	CPU, U2S	ERR_ADDR	(set IADDR)	-	S_RTO	-
P_RDD_REQ	CPU	MEM_ADDR	-	-	S_RBS	(from mem)
	U2S	MEM_ADDR	S_CPD_REQ	P_SACK P_SACKD P_SNACK	S_RBS S_RBS S_RBS	S_CRAB S_CRAB (from mem)
	CPU, U2S	CPU_ADDR	P_RDD_REQ	P_RERRC	S_ERR	-
		SYSIO_ADDR	P_RDD_REQ	P_RERRC	S_ERR	-
		FFB_ADDR	-	-	S_ERR	-
	CPU, U2S	ERR_ADDR	(set IADDR)	-	S_RTO	-
P_WRB_REQ	CPU, U2S	MEM_ADDR	-	-	S_WAB	(to mem)
(data has been invalidated)	CPU	MEM_ADDR	-	-	S_WBCAN	-
	CPU, U2S	SLV_ADDR	UNDEFINED[b]	-	-	-
	CPU, U2S	ERR_ADDR	UNDEFINED[c]	-	-	-
P_WRI_REQ (IVA set)	CPU	MEM_ADDR	S_INV_REQ (to CPU)	P_SACK P_SACKD P_SNACK	S_WAB S_WAB S_WAB	(to mem) (to mem) (to mem)
		SLV_ADDR	P_WRI_REQ	P_WAB	S_WAB	S_SWB
		ERR_ADDR	(set IADDR)	-	S_WAB	-
P_WRI_REQ (IVA not set)	CPU	MEM_ADDR	-	-	S_WAB	(to mem)
		SLV_ADDR	P_WRI_REQ	P_WAB	S_WAB	S_SWB
		ERR_ADDR	9set IADDR)	-	S_WAB	-
P_WRI_REQ (IVA = 'X')	U2S	MEM_ADDR	S_INV_REQ (to CPU)	P_SACK P_SACKD P_SNACK	S_WAB S_WAB S_WAB	(to mem) (to mem) (to mem)
		SLV_ADDR	P_WRI_REQ	P_WAB	S_WAB	S_SWB
		ERR_ADDR	(set IADDR)	-	S_WAB	-

a. The U2S is incapable of generating a P_RDS_REQ of p_RDSA_REQ transaction. If the U2S does generate one of these transactions, the USC will not detect it as an error, and what happens is undefined.

b. The USC does not support cache-able address space at the UPA slaves. If a UPA port issues a P_WRB_REQ to a UPA slave, then a hardware error has occurred since it could not have ever successfully completed a coherent read to that address. However, the USC will not detect it as an error, and what happens is undefined.

c. If a UPA port issues a P_WRB_REQ to an illegal or invalid address, then a hardware error has occurred since it could not have ever successfully completed a coherent read to that address. However, the USC will not detect it as an error, and what happens is undefined.

11

OpenBoot PROM Extension Commands

11.1 Overview

The Themis USP-1 OBP is based on OBP 3.1 from SUN. Themis has added several OBP command extension that are specific to the USP-1: `temp-warining`, `temp-cricitical`, `read-temp`, `vme24-slave-base`, and `vme32-slave-base`.

All other OBP commands are the same as the Sun Ultra I Platform. Specific details of the OBP architecture are defined in the IEEE 1275 specification document.

Reference materials that describe the OBP include:

- OpenBoot 3.1 Command Reference -- Sun Part Number: 802-3242-31
- OpenBoot Quick Reference -- Sun Part Number: 802-5675-31
- Writing FCode 3.1 Programs -- Sun Part Number: 802-3239-31.

11.2 OBP Environment Variables

Table 11-1. **OBP Environment Variables**

Variable	Default	Description
`vme32-slave-base`	0	Used only while OBP is running.
`vme24-slave-base`	0	Used only while OBP is running.
`temp-warning`	53 Degrees C	Shuts down Solaris, if asserted.
`temp-critical`	55 Degrees C	Automatic Hardware Shutdown.
`read-temp`	--	Returns the current value of the thermostat.

These variables may be setup at the **ok** OBP prompt using:

```
setenv variable_name value
```

or when running Solaris with the command:

```
eeprom variable_name=value
```

A board RESET is required for the new values to take effect.

11.3 OBP Environment Variables Description

The following sections describe each USP-1 specific OBP environment variables, their settings, and their usage.

11.3.1 vme32-slave-base

The setting of `vme32-slave-base` is used only while OBP is running. Note that Solaris does not use these values to determine the boards slave addressing.

11.3.2 vme24-slave-base

The setting of `vme24-slave-base` is used only while OBP is running. Note that Solaris does not use these values to determine the boards slave addressing.

11.3.3 temp-warning

If the thermostat temperature exceeds the value held in `temp-warning`, a temperature interrupt will be asserted. Solaris will shutdown when it receives this interrupt.

11.3.4 temp-critical

When the thermostat temperature exceeds the value held in `temp-critical`, the hardware will automatically turn off the clock to the CPU.

OBP will not allow the value of temp-critical in the thermostat to be set to less than 30 degrees C, regardless of the value in the variable `temp-critical`.

Note — The `temp-warning and temp-critical` values are programmed into the thermostat at OBP startup. Newly programmed values for temp-warning and/or temp-critical will not take effect until after a reset.

11.3.5 read-temp (-- temp)

`read-temp` returns the current value of the temperature sensor in degrees C.

11.4 Support Commands and Device Aliases

11.4.1 Support Commands

Several support commands have been added to the USP-1 OBP.

11.4.1.1 probe-scsi2

`probe-scsi2` probes the second scsi bus. This command behaves in a similar fashion to the standard OBP command `probe-scsi`.

11.4.1.2 test net2

`test net2` tests the second ethernet. This command behaves in a similar fashion to the standard OBP command `test net`.

11.4.2 Devices Aliases Under OBP

The device aliases the are same as those used in the Sun Ultra I Platform. Aliases have been added for the scsi2 and net2.

Table 11-2. **OBP Extension Device Alliases**

Device	Alias
devalias scsi2	/sbus/espdma@3,9200000/esp@3,9400000
devalias net2	/sbus/ledma@3,9200010/le@3,9600000
devalias net2-tpe	/sbus/ledma@3,9200010:tpe/le@3,9600000
devalias net2-aui	/sbus/ledma@3,9200010:aui/le@3,9600000

Other device aliases are provided below.

Table 11-3. **OBP Device Aliases**

Device	Alias
devalias keyboard	/sbus/zs@f,1000000
devalias keyboard!	/sbus/zs@f,1000000:forcemode
devalias ttya	/sbus/zs@f,1100000:a
devalias ttyb	/sbus/zs@f,1100000:b
devalias floppy	/sbus/SUNW,fdtwo
devalias scsi	/sbus/espdma@e,8400000/esp@e,8800000

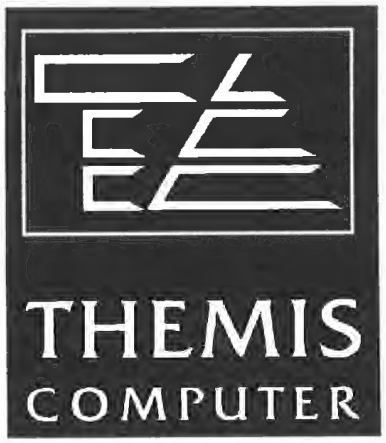
Place
Stamp
Here

Themis Computer

3185 Laurelview Court

Fremont, CA 94538

Attn: Publications Department



Reader Comment Card

We welcome your comments and suggestions to help improve the *USP-1 Programmer's Guide*. Please take time to let us know what you think about these manuals.

- The information provided in the maunal was complete.

  Agree___ Disagree___ Not Applicable___

- The information was well documented and easy to follow.

  Agree___ Disagree___ Not Applicable___

- The information was easily accessible.

  Agree___ Disagree___ Not Applicable___

- The manuals were useful.

  Agree___ Disagree___ Not Applicable___

- Please write down any additional comments you may have about these manuals:

  _____

  _____

  _____

  _____

  _____

  _____

  _____

  _____

Name: _____

Title: _____

Company: _____

Address: _____